Proyecto Fin de Máster

Máster en Ingeniería Electrónica, Robótica y Automática (MIERA)

Sistema de monitorización y control de una instalación acuapónica mediante software OpenSource

Autor: Ricardo García Crespo

Tutores: Ignacio Alvareda

Richard Haes Ellis

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Sistema de monitorización y control de una instalación acuapónica mediante software OpenSource

Autor:
Ricardo García Crespo

Tutores:
Ignacio Alvarado
Richard Haes Ellis

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Máster: Sistema de monitorización y control de una instalación acuapónica mediante software OpenSource

Autor: Ricardo García Crespo

Tutores: Ignacio Alvarado Aldea
Richard Haes Ellis

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

RESUMEN

El proyecto actual consiste en el desarrollo y despliegue de un sistema de monitorización y control de bajo costo para una instalación acuapónica. Es un proyecto Open-Source basado en ESP32 y Raspberry Pi.

Para la medición de los parámetros ambientales y del agua se utilizaron sensores de:

- Temperatura
- pH
- Oxígeno disuelto
- Conductividad eléctrica
- Temperatura ambiental
- Humedad relativa
- Nivel de líquido

Se incluyeron dispositivos adicionales encargados del almacenamiento de datos en local, visualización, temporización y actuación:

- Módulo de memoria MicroSD
- Pantalla LCD
- Reloj en tiempo real
- Relé

El microcontrolador ESP32, junto con los sensores y actuadores distribuidos en la instalación, se encargan de la adquisición de datos y las automatizaciones. Una vez el microcontrolador procesa la información de entrada, ésta es enviada a una Raspberry Pi mediante protocolo MQTT.

La Raspberry recibe la información por MQTT y la almacena en la nube en una base de datos PostgreSQL mediante un script en Python, utilizando para ello la API “psycopg2”.

Para obtener y gestionar una base de datos en la nube se utilizó ElephantSQL, un servicio en la nube que ofrece bases de datos PostgreSQL como servicio (Database-as-a-Service o DBaaS).

Con respecto al hardware y conexiones, para las primeras pruebas se utilizó una protoboard. La segunda etapa consistió en crear una solución más robusto y estable, soldando todos los componentes y dispositivos en una placa perforada. Una vez validado todo el software se diseñó y mandó a fabricar una placa de circuito impreso (PCB) como prototipo definitivo.

Los softwares utilizados fueron: EasyEDA, ArduinoIDE, VSCode, Geany, ElephantSQL, PgAdmin4, VNC Viewer, RaspberryPi Imager y Putty.

Los lenguajes de programación utilizados fueron: Arduino (C++), Python y SQL.

ABSTRACT

The current project consist of the development and deployment of a low cost monitoring and control system for an aquaponic installation. It is an Open-Source Project based on ESP32 and Raspberry Pi.

For the measurement of environmental and water parameters, sensors of:

- Temperature
- pH
- Dissolved oxygen
- Electric conductivity
- Environmental temperature
- Relative humidity
- Liquid level

Additional devices responsable for local data storage, visualization, timing and actuation were included:

- MicroSD memory module
- Display LCD
- Real time clock
- Relay

The ESP32 microcontroller, together with the sensors and actuators distributed in the installation, are responsable for data acquisition and automation. Once the microcrontroller precesses the input information, it is sent to a Raspberry Pi by MQTT protocol.

The Raspberry receives the information via MQTT and stores it in the cloud in a PostgreSQL database by means of a Python script, using the “psycopg2” API.

To obtain and manage a database in the cloud, ElephantSQL was used, a cloud service that offers PostgreSQL databases as a service (Database-as-a-Service or DBaaS).

Regarding the hardware and connections, for the first tests a breadboard was used. The second stage consisted of creating something more robust and stable, soldering all the components and devices on a perforated plate. Once all the software was validated, a printed circuit board (PCB) was designed and manufacturated as a final prototype.

The software used were: EasyEDA, ArduinoIDE, VSCode, Geany, ElephantSQL, PgAdmin4, VNCViewer, RaspberryPi Imager and Putty.
The programming languages used were: Arduino (C++), Python and SQL.

ÍNDICE DE CONTENIDOS

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

1.1. Contexto

Promover el uso de sistemas acuapónicos hoy en día es crucial para abordar desafíos tales como:

- La seguridad alimentaria
- La conservación de recursos
- La reducción del impacto ambiental
- La innovación tecnológica

Estos sistemas ofrecen una forma sostenible y eficiente de producir alimentos frescos y nutritivos, especialmente en áreas urbanas y comunidades con recursos limitados.

Sin embargo, construir este tipo de instalaciones presenta problemas debido a los altos costos asociados a los materiales, equipos y sistemas existentes, dificultando la implantación de estas técnicas en ciertas áreas.

Además, la complejidad de la integración y la falta de soluciones específicas para acuaponía limitan la eficiencia y la precisión de estos sistemas.

Por ello, es fundamental desarrollar soluciones de bajo costo y accesibles que aborden las necesidades específicas de los sistemas acuapónicos, permitiendo que agricultores, emprendedores y comunidades de vecinos puedan implementar estos sistemas con facilidad y obteniendo buenos resultados.

1.2. Objetivo

Este proyecto tiene como objetivo el desarrollo y despliegue de un sistema de monitorización y control de bajo costo para una instalación acuapónica, almacenando la información en una base de datos en la nube.

2 MARCO TEÓRICO

2.1. Acuaponía

2.1.1 Definición

El término “acuaponía” proviene de la combinación del término acuicultura e hidroponía.

La acuicultura es el conjunto de actividades, técnicas y conocimientos relacionados con la crianza de especies acuáticas, tanto animales como vegetales.

Figura 1 – Tanque de peces en instalación acuícola

Por otro lado, la hidroponía es una técnica de cultivo de plantas en la que se utilizan soluciones nutritivas, en lugar de suelo agrícola, que proporcionan los nutrientes necesarios para el crecimiento y desarrollo de las plantas. En lugar de crecer en tierra, las raíces se sumergen en una solución de agua y nutrientes ajustados cuidadosamente en función de las necesidades del cultivo. Se utiliza comúnmente para el cultivo en ambientes controlados, como en invernaderos o interiores, permitiendo un mayor control sobre el entorno de crecimiento.

Figura 2 – Cultivo hidropónico de lechuga

Combinando dichos conceptos, la acuicultura es la técnica de cultivo de plantas y peces que combina la acuicultura tradicional con la hidroponía mediante el uso de sistemas de recirculación de agua y nutrientes en un ambiente controlado.

2.1.2 Principios

Concibiendo un sistema acuapónico como un modelo de producción, el alimento suministrado a los peces funciona como la entrada de energía al sistema, del que obtienen los nutrientes necesarios para el desarrollo de sus vidas. Los peces absorben del alimento lo necesario, excretando aquello que son producto o desecho de su metabolismo.

Tanto los desechos como el alimento no consumido se convierten en el sustrato del que se alimentan las bacterias nitrificantes, las cuales se encargan de convertir los residuos tóxicos, como el amoníaco, en nitritos y posteriormente en nitratos.

Los nitratos disueltos en el agua son absorbidos por las raíces de las pantas que los usan como fuente de nutrientes. A medida que el agua atraviesa el cultivo hidropónico, las raíces de las plantas y los microorganismos asociados a ellas eliminan diversos contaminantes, como metales pesados y contaminantes orgánicos. También se libera oxígeno al agua mediante la fotosíntesis, beneficiando todo ello a los peces y demás organismos acuáticos.

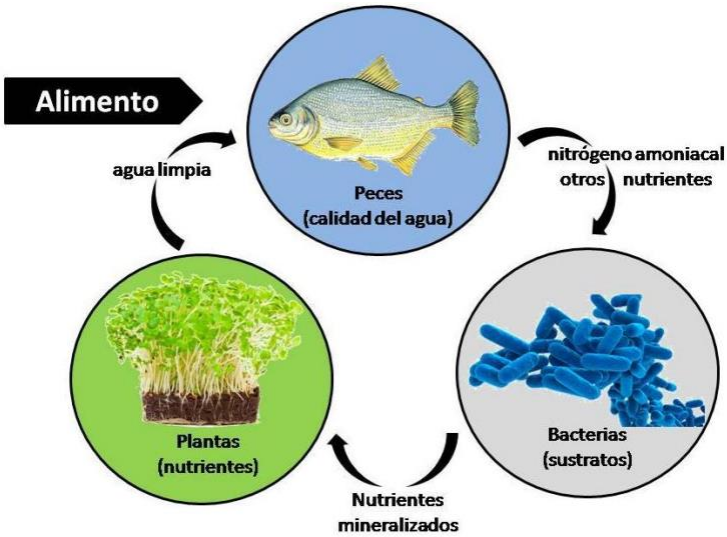


Figura 3 – Ciclo biológico de la acuaponía

Un sistema acuapónico está formado por diferentes tipos de especies que trabajan de manera cooperativa y sostenible para autoabastecerse.

El control de los parámetros ambientales, y sobre todo del agua, es crucial en este tipo de instalaciones, lo que hace necesario implementar precisos sistemas de monitorización y control que garanticen unas determinadas condiciones para el agua en cada una de las etapas.

2.1.3 Importancia

Como modelo de producción sostenible de alimentos, la acuaponía posee muchas ventajas beneficiando tanto al ser humano como al planeta por numerosos aspectos:

- Producción sostenible de alimentos
- Reducción de la huella de carbono
- Seguridad alimentaria
- Diversificación de la producción alimentaria
- Rentabilidad económica
- Investigación y desarrollo

Se exponen los beneficios que portan este tipo de metodologías de cultivo atendiendo a tres aspectos principales: social, medioambiental y económico.

2.1.3.1 Importancia social

El desarrollo de sistemas acuapónicos tiene un impacto social positivo por diferentes motivos:

1. Proporcionan una fuente de alimentos frescos y saludables a comunidades locales.
2. Permiten el cultivo de alimentos vegetales y animales en áreas tanto urbanas como rurales, mejorando la seguridad alimentaria y reduciendo la dependencia de alimentos importados.
3. Pueden ser la solución para aquellos lugares donde no hay suficiente tierra o agua para cultivos tradicionales.
4. Pueden ser utilizados para abordar problemas sociales como la inseguridad alimentaria y la pobreza, proporcionando alimentos frescos y saludables a personas de bajos ingresos que de otra manera tendrían acceso limitado a estos alimentos.
5. Tienen la capacidad de mejorar la resiliencia de las comunidades antes desastres naturales y otros eventos imprevistos. Debido a que utilizan menos recursos naturales y son más resistentes a las condiciones climáticas adversas que los cultivos tradicionales, pueden seguir produciendo alimentos incluso en condiciones difíciles.
6. Pueden ser utilizados como herramienta educativa para enseñar a las personas sobre producción alimentaria y agricultura sostenible.

2.1.3.2 Importancia mediombiental

Con respecto al medioambiente, el desarrollo de este tipo de sistemas también conlleva numerosas ventajas:

1. Promueven un uso eficiente del agua, ya que necesitan una cantidad significativamente menor que los cultivos tradicionales. Conlleva una reducción de la demanda de agua dulce, disminuyendo la presión sobre los recursos hídricos del planeta.
2. Reducción de la huella de carbono mediante la reducción de residuos y contaminantes generados. Se elimina la necesidad de utilizar fertilizantes y pesticidas químicos típicos de la agricultura tradicional.
   Además, la producción local de alimentos puede reducir la necesidad de trasporte y

almacenamiento de alimentos, lo que a su vez reduce las emisiones de gases de efecto invernadero asociadas.

3. Se promueve la biodiversidad y la restauración de los ecosistemas acuáticos. Al cultivar peces y plantas en un entorno controlado y sostenible, los sistemas acuapónicos pueden ayudar a proteger las especies de peces y plantas en peligro de extinción, así como restaurar los ecosistemas acuáticos degradados.

2.1.3.3 Importancia económica

Como no podía ser menos, la acuaponía también tiene una gran importancia a nivel económico para agricultores, empresas y comunidades que utilizan estos sistemas de producción.

1. Permite generar empleo local en áreas urbanas y rurales, reduciendo la necesidad de importar alimentos, beneficiando así la economía regional.
2. Aumenta la rentabilidad de las explotaciones agrícolas.
   Permite la producción de dos tipos de productos (vegetales y peces) compartiendo el agua, la energía y el alimento que necesitan, eliminando la necesidad de utilizar fertilizantes químicos y pesticidas. Todo esto conlleva una reducción del coste de producción y un aumento de los márgenes de beneficio de la explotación.
3. Posibilita la producción alimentaria durante todo el año, independientemente de las condiciones climáticas exteriores, lo que puede aumentar la producción y los ingresos en ciertas regiones.

2.1.4 Elementos de una instalación acuapónica

Una instalación acuapónica consta de varios subsistemas que trabajan en conjunto para crear un sistema de cultivo integrado y sostenible. A continuación se describen cada uno de ellos:

2.1.4.1 Tanque de peces

Es el componente principal de la acuicultura en el sistema acuapónico, ya que es el lugar donde se crían los peces y se acumulan los desechos orgánicos. Pueden estar fabricados de una gran variedad de materiales, dependiendo del presupuesto, durabilidad requerida, disponibildad y otros factores. Normalmente suelen ser de:

- Plástico (PVC o polietileno)
- Fibra de vidrio
- Acero inoxidable

2.1.4.2 Filtro mecánico

Encargado de eliminar los sólidos en suspensión y otras partículas grandes que puedan estar presentes en el tanque de peces antes de que el agua sea bombeada al sistema de cultivo hidropónico.

2.1.4.3 Filtro biológico

Encargado de realizar el proceso de nitrificación. En él, las bacterias nitrificantes convierten los desechos orgánicos de los peces en compuestos nitrogenados que las plantas usan como fuente de nutrientes.

2.1.4.4 Sistema de cultivo hidropónico

Consiste en un lecho de cultivo en el que se colocan las plantas. Las raíces se sumergen en el agua que fluye a través del sistema de cultivo, absorbiendo los nutrientes necesarios para su crecimiento.

2.1.4.5 Sistema de bombeo y tuberías

La bomba y las tuberías se utilizan para transportar el agua desde el tanque de peces, a través de los filtros, hasta el cultivo hidropónico, devolviéndola posteriormente limpia al tanque de peces.

2.1.4.6 Sistema de control

Conjunto de equipos y dispositivos encargados de la monitorización y control de la instalación. Normalmente consta de:

- Sensores: Encargados de conocer los parámetros ambientales y del agua.
- Actuadores: Encargados de realizar acciones sobre la instalación con el fin de modificar su comportamiento, guiando al sistema hacia un estado óptimo.
- Controlador: Cerebro del sistema de control encargado de monitorizar la instalación y accionar los actuadores en función de las lecturas de los sensores.

2.1.5 Control de los parámetros del agua

En un sistema acuapónico el agua es el elemento principal que permite el desarrollo de las especies que se cultivan, ya que:

- Actúa como un medio de trasporte de nutrientes.
- Aporta oxígeno para peces y plantas.
- Actúa como un medio de limpieza, llevándose consigo todas las sustancias que se desechan en cada una de las etapas.
- Ayuda a mantener la estabilidad del sistema.

La calidad del agua es un factor crítico, por ese motivo es crucial controlar los parámetros cuidadosamente en función de las necesidades de las especies que habitan en el sistema. Los parámetros principales a controlar en todo sistema acuapónico son:

2.1.5.1 Temperatura del agua

La temperatura es un factor crítico que afecta directamente a la tasa de crecimiento de los peces y plantas.

La temperatura del agua se reduce demasiado, disminuye el metabolismo de los peces, reduciendo su capacidad para digerir los alimentos y afectando directamente a la tasa de

crecimiento.

La temperatura también afecta a la tasa de nitrificación. Las bacterias nitrificantes operan mejor a temperaturas cálidas, mientras que a temperaturas bajas pueden disminuir su actividad reduciendo la capacidad del sistema para procesar los desechos de los peces.

En este tipo de sistemas las plantas también necesitan una temperatura cálida para desarrollarse correctamente, aunque si aumenta demasiado puede afectar negativamente a la absorción de nutrientes y aumentar las posibilidades de enfermedades.

La mayoría de especies que se cultivan en este tipo de sistemas prefieren una temperatura entre 20 y 30 grados Celsius.

2.1.5.2 PH

El pH es otro factor crítico que afecta directamente la capacidad de los peces, plantas y bacterias para funcionar correctamente en el sistema, afectando directamente a la toxicidad de los nutrientes en el agua.

El pH es una medida de concentración de iones H+ en solución, definiéndose de la siguiente manera:

$$pH = -\log([H_+])$$

Las bacterias nitrificantes que convierten el amoníaco en nitratos son sensibles al pH del agua, funcionando mejor en un rango de 6.5 a 8. Una fluctuación significativa en los niveles de pH puede afectar negativamente a la capacidad para procesar los desechos de los peces.

También afecta a la capacidad de las plantas para absorber los nutrientes.

Con respecto a los peces, un pH bajo (ácido) puede provocar irritación en las branquias, mientras que uno alto (básico) puede reducir su capacidad para respirar.

Mantener un pH equilibrado es crucial para garantizar que los nutrientes sean utilizados de forma segura y efectiva para los organismos del sistema.

2.1.5.3 Oxígeno disuelto

Todos los organismos en el sistema acuapónico necesitan oxígeno para respirar.

Si la concentración de oxígeno disminuye por debajo de un nivel crítico, los peces pueden experimentar estrés, enfermedades y una reducción de su capacidad para crecer y reproducirse. Las bacterias nitrificantes pueden reducir su actividad o morir, conllevando un aumento en la acumulación de amoníaco del sistema. La capacidad de absorción de nutrientes de las plantas disminuye, afectando directamente a su crecimiento y a su capacidad de producir cosechas.

2.1.5.4 Alcalinidad y dureza

La alcalinidad o basicidad del agua es la medida de su capacidad para neutralizar ácidos, mientras que la dureza se refiere a la cantidad de calcio y magnesio disueltos en ella.

Estos minerales son necesarios para el desarrollo de las plantas y peces. Si la dureza del agua es demasiado baja, las plantas y peces pueden sufrir deficiencias de calcio y

magnesio, afectando negativamente a su salud y crecimiento. En cambio, si la dureza si demasiado alta pueden formarse depósitos de calcio y magnesio en el sistema, acarreando obstrucciones en los conductos y afectado a la eficacia de los sistemas de filtración.

Los niveles adecuados de alcalinidad y dureza pueden variar según las especies de plantas y peces que se cultivan, por lo que es importante investigar las necesidades específicas de las especies que se cultivan.

2.1.5.5 Conductividad eléctrica

La conductividad eléctrica del agua es un parámetro importante a ser controlado en toda instalación acuapónica, ya que está directamente relacionado con la cantidad de sales disueltas, es decir, con la cantidad de nutrientes. Una concentración adecuada de nutrientes es esencial para que las plantas crezcan saludables y asegurar que los peces tengan un ambiente adecuado para vivir.

Si la conductividad eléctrica es demasiado alta puede indicar que existe una acumulación excesiva de nutrientes en el agua, volviéndose tóxicos para los peces y otros organismos. Además puede promover el crecimiento de algas y otros microorganismos no deseados, obstruyendo los conductos y afectado a la eficacia de los sistemas de filtración.

Por el contrario, si la conductividad eléctrica es demasiado baja puede indicar que la solución acuosa carece de nutrientes para las plantas, provocando una menor producción de alimentos y reduciendo su capacidad para purificar el agua.

2.2. Sistemas de monitorización y control

2.2.1. Dispositivos de medición

Una vez descrita la importancia de los diferentes parámetros sobre la calidad del agua, queda clara la necesidad de controlarlos de manera precisa con el fin de mantener a las diferentes especies en condiciones saludables, optimizando la producción y aumentando la vida útil de los equipos.

Para ello es necesario incluir sensores de:

- Temperatura del agua
- pH
- Oxígeno disuelto
- Conductividad eléctrica
- Nivel de agua en tanques

También es conveniente incluir sensores ambientales para conocer la temperatura y la humedad relativa existente en el entorno de la instalación.

- Temperatura ambiente
- Humedad relativa ambiente

Para cada uno de los parámetros anteriores se describen los sensores más utilizados en acuaponía actualmente.

2.2.1.1. Sensores de temperatura

Termistor

Es un tipo de sensor de temperatura que funciona mediante la variación no lineal de la resistencia eléctrica de un material conductor. Existendos dos tipos principales:

- Termistor NTC (Negative Temperature Coefficient): Disminuyen su resistencia eléctrica a medida que aumenta la temperatura.
- Termistor PTC (Positive Temperature Coefficient): Aumentan su resistencia eléctrica a medida que aumenta la temperatura.

RTD (Resistive Temperature Device)

Es un tipo de sensor de temperatura que utiliza la variación lineal de la resistencia eléctrica de un material conductor para medir la temperatura. El material que más se suele utilizar es el platino (PT100), aunque también se utilizan otros materiales como cobre y níquel.

Termopar

Es un tipo de sensor de temperatura basado en el efecto Seebeck. Cuando dos metales diferentes se unen por sus extremos en un circuito cerrado y se someten a una variación de temperatura, se produce una pequeña diferencia de tensión entre sus extremos, la cual puede medirse y convertirse en una medida de temperatura.

Existen varios tipos de termopares, que se diferencian por los materiales utilizados en la unión de medición y por sus características de temperatura y sensibilidad (termopar tipo K, J, T, E, S, y B).

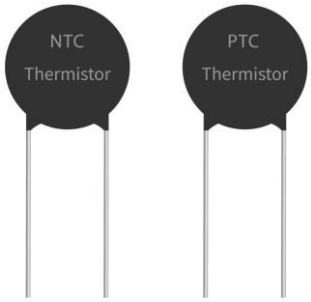


Figura 4 – Termistor NTC y PTC

Figura 5 – Sensor de temperatura RTD

Figura 6 – Termopar tipo K

Los termopares tienen un costo mayor y son más utilizados en entornos industriales. En cambio, existen multitud de termistores en el mercado para medir temperatura de líquidos en aplicaciones de bajo costo. Una alternativa muy utilizada es el DS18B20.

2.2.1.2. Sensores de pH

Sensores de electrodo vidrio

Son uno de los sensores mas utilizados para la medición del pH en acuaponía. Utilizan electrodo de vidrio sensible a los iones hidrógeno de una solución, generando una señal eléctrica proporcional al nivel de pH de ésta. Son sensores altamente precisos y confiables, aunque requieren un mantenimiento regular y una calibración adecuada.

Sensores de ion selectivo (ISE)

Utilizan una membrana selectiva que permite el paso de los iones de hidrógeno presentes en una solución, generando una señal eléctrica proporcional a la concentración de dichos iones, lo que se traduce en una medida de pH. Son sensores altamente sensibles y precisos, aunque suelen ser más costosos y requieren una calibración más regular.

Sensores de película delgada

Este tipo de sensores utilizan una capa delgada de material sensible al pH para medir la acidez o alcalinidad del agua. La película delgada sensible al pH se aplica en un electrodo y se sumerge en el agua a analizar. Al interactuar ocurre una reacción química que produce un cambio de color o un cambio en la respuesta óptica del material, midiéndolo mediante un fotodiodo u otro dispositivo similar, conviertiendolo en una medida del pH.

Sensores de esmalte cerámico

Son un tipo de sensor de pH que utilizan un esmalte cerámico como elemento sensible. Cuando el esmalte cerámico entra en contacto con una solución, los iones hidrógeno de la solución reaccionan con los iones del esmalte cerámico, generando una carga eléctrica detactada como voltaje. Dicho voltaje se convierte posteriormente en una medida de pH.

Figura 7 – Sensor de pH de electrodo de vidrio

Figura 8 – Sensor de pH de ión selectivo (ISE)

Figura 9 – Sensor de pH de película delgada

Figura 10 – Sensor de pH de esmalte cerámico

2.2.1.3. Sensores de oxígeno disuelto

Sensores electroquímicos

Utilizan una celda electroquímica para medir el oxígeno disuelto en el agua. La celda contiene un electrodo de trabajo y un electrodo de referencia sumergidos en la solución. La diferencia de potencial generada por la reacción electroquímica es medida y es convertida en una lectura de concentración de oxígeno disuelto.

Sensores ópticos

Sensor capaz de medir la proporción de oxígeno disuelto en una solución añadiendo a ésta una sustancia fluorescente que absorbe luz y emite luz a longitudes de onda diferentes. Al lanzar un haz de luz sobre la solución que contiene la sustancia fluorescente, se mide la cantidad de luz emitida por la sustancia, siendo proporcional a la cantidad de oxígeo disuelto que contiene la solución.

Figura 11 – Sensor electroquímico de oxígeno disuelto

Figura 12 – Sensor óptico de oxígeno disuelto

2.2.1.4. Sensores de conductividad eléctrica

Sensor inductivo

Es capaz de medir la conductividad eléctrica de un líquido mediante la inducción de un campo electromágnético sobre él. Su principio de funcionamiento se basa en la Ley de Faraday de inducción electromagnética.

El sensor consta de una bobina que se coloca alrededor de un tubo que contiene la solución. Al aplicar una corriente alterna sobre la bobina se genera un campo electromagnético en el líquido que induce una corriente eléctrica en el mismo. La magnitud de dicha corriente es proporcional a la conductividad eléctrica de la solución. El sensor mide la impedancia de la bobina, influenciada por la corriente eléctrica inducida, convertiéndola en una lectura de conductividad.

Sensor de contacto

Sensor de conductividad eléctrica de contacto que utiliza dos electrodos sumergidos en la

solución para medir la conductividad eléctrica de ésta. Funciona en base a la Ley de Ohm, que establece que la corriente eléctrica que fluye a través de un conductor es directamente proporcional al voltaje aplicado e inversamente proporcional a la resistencia del material.

La conductivdad eléctrica se calcula a partir de la Resistencia eléctrica medida y otros factores, como la geometría del sensor y la temperature de la solución.

Figura 13 – Sensor de conductividad eléctrica inductivo

Figura 14 – Sensor de conductividad eléctrica de contacto

2.2.1.5. Sensores de nivel

Existen numerosos tipos de sensores utilizados para medir el nivel de líquido en tanques, aunque en este caso sólo se describe un tipo de ellos, siendo la alternativa más simple y económica del mercado:

Sensores flotador

Es un sensor binario ON-OFF (detector) que mide el nivel de líquido en un tanque mediante el uso de un flotador que se mueve hacia arriba o abajo cuando el nivel del líquido lo alcanza.

Figura 15 – Sensor de nivel tipo flotador (float switch)

2.2.1.6. Sensores de humedad y temperature ambiente

A continuación se presentan los sensores temperatura y humedad ambiente más conocidos y económicos del mercado:

DHT22

Sensor digital de temperatura y humedad relativa de bajo costo. Está formado por un sensor capacitivo para la medida de humedad y un termistor para la medida de temperatura.

BME280

Sensor digital de temperatura, humedad relativa y presión atmosférica de alta precisión. Utiliza un sensor capacitivo para medir la humedad, un termistor para medir la temperatura y un sensor de presión piezoeléctrico para medir la presión attmosférica.

Figura 16 – Sensor ambiental DHT22

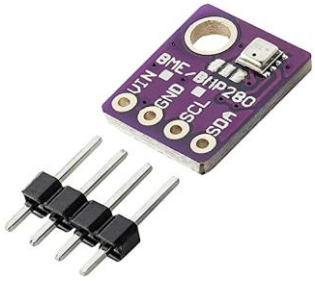


Figura 17- Sensor ambiental BME280

2.2.1.7. Comparativa

Parámetro	Sensor	Salida	Rango de medida	Precisión	Consumo Voltaje	Consumo Corriente
Temperatura	Termistor	Analog	-40 a 160C	+/- 0.1C	máx 6 VDC	máx 450uA
	RTD	Analog	-190 a 260C	+/- 0.1 a +/- 0.5C	máx 5 VDC	máx 5mA
	Termopar K	Analog	-200 a 1350C	+/- 1.4 a +/- 4C	0 VDC	0A
pH	De vidrio	Analog	0 a 14	+/- 0.1	máx 12 VDC	máx 10mA
	ISE	Digital	0-14	+/- 0.05	máx 5 VDC	máx 0.1A
	De esmalte	Analog	0-14	+/- 0.2	máx 24 VDC	máx 0.5mA
Oxígeno disuelto	Electroquímico	Analog	0-100 mg/L	+/+ 0.05 mg/L	máx 12 VDC	máx 50mA

	Óptico	Analog	0-20 mg/L	+/+ 0.05 mg/L	máx 24 VDC	máx 100mA
Conductividad eléctrica	Inductivo	Analog	0 a 20 mS/cm o 200 a 2000 mS/cm	+/- 1-2% lectura	máx 24 VDC	máx 500mA
	De contacto	Analog	0 a 2000 mS/cm	+/- 5% lectura	máx 12 VDC	
Nivel (distancia)	Flotador	Digital	---	---	---	---
Temperatura y humedad ambiente	DHT22	Digital	-40 a 80 C 0 a 100% HR	+/- 0.5 C +/- 0.1% HR	5.5 VDC	2.5 VDC
	BME280	Digital	-40 a 85 C 0 a 100% HR	+/- 1 C +/- 3% HR	3.6 VDC	1.8 mA

Tabla 1 - Comparativa de sensores

2.2.2. Dispositivos de actuación

En instalaciones acuapónicas los dispositivos de actuación son necesarios para automatizar y controlar el sistema, lo que permite un mejor rendimiento y un ahorro de tiempo y esfuerzo. Estos dispositivos se utilizan principalmente para controlar el flujo del agua, el suministro de nutrientes y la iluminación.

Los dispositivos de actuación más comunes en instalaciones acuapónicas son los siguientes.

2.2.2.1. Bombas de agua

Utilizada para transportar el agua desde el tanque de peces hasta el cultivo hidropónico y viceversa.

Bomba centrífuga

Bomba hidráulica que funciona mediante la fuerza centrífuga generada por un impulsor que gira dentro de una carcasa, empujando el agua hacia el conducto de salida. El impulsor está montado en un eje conectado a un motor eléctrico que proporciona la energía necesaria para hacerlo girar.

Bomba de desplazamiento positivo

Bomba hidráulica que se caracteriza por mover el fluido a través de un mecanismo que desplaza un volumen fijo de líquido en cada ciclo o rotación.

Las bombas peristálticas son un tipo de bomba de desplazamiento positivo muy utilizada en instalaciones acuapónicas ya que permiten un ajuste fino del caudal entregado. Permiten añadir cantidades concretas de determinadas sustancias, como soluciones correctoras de

pH en instalaciones acuapónicas.

Figura 18 – Bomba centrífuga

Figura 19 – Bomba peristáltica

2.2.2.2. Electroválvulas

Las electroválvulas son componentes esenciales en instalaciones acuapónicas. Permiten el corte y la apertura del suministro de agua en áreas específicas, lo que es importante para el control y regulación del sistema.

Pueden controlarse mediante temporizadores o controladores de diferentes tipos (de pH, de nivel, de temperatura,…). Existen diferentes tipos de electroválvulas en el mercado, siendo las más communes:

- Solenoide
- De compuerta
- De bola
- De aguja

Figura 20 – Electroválvula neumática

2.2.3. Controlador

Es necesario inlcluir un dispositvo que se encargue de controlar y automatizar de manera precisa y eficiente los parámetros y procesos críticos en el sistema, aumentando la productividad y rentabilidad.

Para ello se utilizan microcontroladores, cuya función será adquirir e interpreter las lecturas de los sensores, activando o desactivando los elementos de actuación de manera lógica en función de las necesidades del sistema.

También pueden encargarse de almacenar la información del sistema en servidores o bases de datos.

Existen multitud de microcontroladores en el mercado, aunque aquí se describen los más comunes actualmente.

2.2.3.1. Arduino

Arduino es una plataforma de desarrollo de hardware y software libre que permite crear prototipos de dispositivos electrónicos de manera fácil y accesible. Se compone de una serie de placas hardware y un entorno de programación integrado propio (ArduinoIDE).

Existen numerosas de placas de Arduino, aunque aquí se muestran las más conocidas:

- Arduino Uno
- Arduino Mega
- Arduino Nano
- Arduino Due

Figura 21 – Arduino Uno

Figura 22 – Arduino Due

Figura 23 – Arduino Nano

Figura 24 – Arduino Mega

2.2.3.2. Espressif

Espressif Systems es una empresa china especializada en el diseño y fabricación de microcontroladores, sistemas en chip (SoC) y módulos de comunicación inalámbrica. Posee un conjunto de herramientas de software para el desarrollo de aplicaciones con microcontrolador llamado ESP-IDF (Espressif IoT Development Framework). Los microcontroladores suyos más conocidos son:

- ESP8266
- ESP32

Figura 25 – ESP8266

Figura 26 – ESP32

2.2.3.3. Raspberry Pi

Raspberry Pi es una serie de ordenadores de placa reducida (no microcontroladores) de bajo costo y reducido tamaño desarrolladas por la Fundación Raspberry Pi. Son ordenadores muy versátiles y se utilizan para una amplia variedad de proyectos, desde servidores web hasta sistemas de automatización, robótica, educación, entretenimiento, etc. Existen diferentes versiones de la placa, siendo la más popular actualmente la Raspberry Pi 4 B:

- Raspberry Pi 1, 2, 3 y 4
- Raspberry Pi Zero

Figura 27 – Raspberry Pi 4B

Figura 28 – Raspberry Pi Zero

2.2.3.4. Comparativa

Placa	**Microcontrolador**	**Voltaje de operación**	**Memoria flash**	**Memoria RAM**	**Velocidad reloj**	**E/S digitales**	**Comunicación**
Arduino UNO	ATmega328P	5 VDC	32 KB	2 KB	16 MHz	14	UART, I2C, SPI, USB
Arduino MEGA	ATmega2560	5 VDC	256 KB	8 KB	16 MHz	54	UART, I2C, SPI, USB
ESP3266	ESP8266	3.3 VDC	4 MB	80 KB	80/160 MHz	17	UART, I2C, SPI, Wi-Fi
ESP32	ESP32	3.3 VDC	4 MB	520 KB	80/240 MHz	34	UART, I2C, SPI, Wi-Fi, Bluetooth
Raspberry Pi 4B	Broadcom BCM2711, quad-core Cortex-A72	5 VDC	---	4 GB	1.5 GHz	26	UART, I2C, SPI, USB, Wi-Fi, Ethernet, Bluetooth

Tabla 2 – Comparativa de placas de desarrollo

2.3. Bases de datos

2.3.1. Definición

Conjunto de datos organizados y estructurados que pertenecen a un mismo contexto y se utilizan para administrar grandes cantidades de información electrónicamente. Dichos datos son agrupados en dispositivos de almacenamiento, como discos duros o servidores. No solo se encargan del almacenamiento, sino también de conectar los datos entre sí en una unidad lógica.

Dentro de las bases de datos la información se organiza en tablas que contienen filas y columnas, y pueden ser manipuladas mediante operaciones como inserción, actualización, búsqueda y eliminación.

Existen sistemas gestores de bases de datos (DBMS), que son softwares utilizados para administrarlas y gestionarlas. Proporcionan una interfaz cómoda para crear, almacenar, actualizar, recuperar o eliminar información en bases de datos, garantizando la integridad, seguridad y consistencia de éstos.

2.3.2. Necesidad

En instalaciones acuapónicas es fundamental incluir el uso de una base de datos para mantener un registro del estado del sistema.

Permite almacenar la información obtenida por el sistema de control, como las lecturas de los sensores, el registro de eventos o el historial de operaciones. Esto permite obtener un registro completo de la operación del sistema, ayudando a identificar patrones y tendencias que puedan ayudar a detector errores, carencias o mejorar le eficiencia y rendimiento del sistema.

2.3.3. Alternativas Open-Source

En este apartado se exponen y comparan las distintas alternativas para bases de datos gratuitas que existen en el mercado actualmente.

2.3.3.1. MySQL

MySQL es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto muy popular. Fue desarrollado por Oracle Corporation y se distribuye bajo una licencia de software libre. Es compatible con una amplia gama de sistemas operativos, como Windows, Linux, macOS y Solaris, entre otros.

Utiliza el lenguaje SQL (Structured Query Language) para acceder y manipular los datos almacenados en la base de datos.

Figura 29 – Base de datos MySQL

2.3.3.2. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto desarrollado por una comunidad de desarrolladores de todo el mundo. Fue creado por Michael Stonebraker en la Universidad de California de Berkeley en la década de los 80, y desde entonces se ha convertido en una de las bases de datos más populares en el mundo empresarial.

Al igual que MySQL, es compatible con una amplia gama de sistemas operativos, utilizando el lenguaje SQL. Además ofrece un amplio conjunto de carcaterísticas avanzadas que lo hacen adecuado para proyectos de mayor complejidad.

Figura 30 – Base de datos PostgreSQL

2.3.3.3. MongoDB

MongoDB es un Sistema de gestión de bases de datos NoSQL de código abierto desarrollado por MongoDB Inc. Es una base de datos orientada a documentos, lo que es ideal para aplicaciones con datos no estructurados o semiestructurados.

Compatable también con una alta gana de sistemas operativos, utiliza el format BSON (Binary JSON), utilizado para el intercambio de datos entre aplicaciones.

Figura 31- Base de datos MongoDB

2.3.3.4. SQLite

SQLite es un Sistema de gestión de bases de datos relacional de código abierto que se ejecuta en la mayoría de sistemas operativos (Windows, Linux, MacOS, iOS, Android). A diferencia de otros sistemas de gestión de bases de datos, SQLite se implementa como una biblioteca de enlace dinámico integrada directamente en la aplicación que utiliza la base de datos. Es liviano y rápido, lo que lo hace ideal para aplicaciones que requieran poca memoria y alta velocidad de acceso a los datos.

Es fácil de usar y no require un servidor separado, sino que la base de datos se almacena en un archivo local, facilitanto la portabilidad y el intercambio de datos entre aplicaciones.

Figura 32 - Base de datos SQLite

2.3.3.5. Comparativa

A continuación se muestra una tabla comparativa de las bases de datos descritas anteriormente, atendiendo a criterios como el tipo, el lenguaje utilizado, la escalablidad, la seguridad, la flexibilidad y la portabilidad.

Base de datos	Tipo	Lenguaje	Escalabilidad	Seguridad	Flexibilidad	Portabilidad
MySQL	Relacional	SQL	Alta	Buena	Media	Buena
PostgreSQL	Relacional	SQL	Alta	Muy buena	Media	Buena
MongoDB	NoSQL	BSON/JSON	Alta	Media	Alta	Media
SQLite	Relacional	SQL	Media	Buena	Baja	Muy buena

Tabla 3 - Comparativa de bases de datos Open-Source

3 DESCRIPCIÓN DEL SISTEMA

En este tercer apartado se describen todos los elementos que forman el sistema de monitorización y control acuapónico. Se expone una vision general de la instalación, indicando que dispositivos se integraron y el flujo de información entre ellos. Posteriormente se incluye una descripción más detallada de cada uno, además de las herramientas software utilizadas.

3.1. Visión general

El sistema desarrollado consta de los dispositivos mencionados en la siguiente tabla:

Tipo de dispositivo	**Dispositivo**	**Modelo**	**Unidades**
Sensores	Sensor de temperatura del agua	DS18B20	2
	Sensor de pH	DFRobot SEN0169	1
	Sensor de oxígeno disuelto	DFRobot SEN0237	1
	Sensor de conductividad eléctrica	DFR0300	1
	Sensor de nivel	Interruptor flotador RS PRO	2
	Sensor de temperatura y humedad ambiental	DHT22	2
Microcontroladores	ESP32	ESP32 FireBeetle Board	1
	Raspberry Pi	Raspberry Pi 4 B	1
Otros dispositivos	Módulo SD	DFRobot MicroSD Module V1.0	1
	Reloj RTC	RTC DS3231	1
	Display LCD	LCD I2C 2004A-V1.1	1
	Relé	Songle 2 relay module	2

Tabla 4 – Dispositivos seleccionados para el prototipo AQUACOL

El esquema presentado muestra el flujo de información del sistema, desde que el ESP32 adquiere las lecturas de la instalación hasta que éstas se almacenan en la base de datos PostgreSQL, visualizándose mediante herramientas como PgAdmin4 o ElephantSQL.

Dado que se presentaron algunos impedimentos para comunicar directamente el ESP32 con la base de datos, se optó por utlizar una Raspberry Pi encargada de recoger la información de la instalación e insertarla en la base de datos mediante un script en Python, utilizando para ello la API “psycopg2”.

La comunicación entre ESP32 y Raspberry Pi se realizó mediante protocolo inalámbrico MQTT, instalando el conocido broker “Mosquitto” de manera local en la Raspberry Pi.

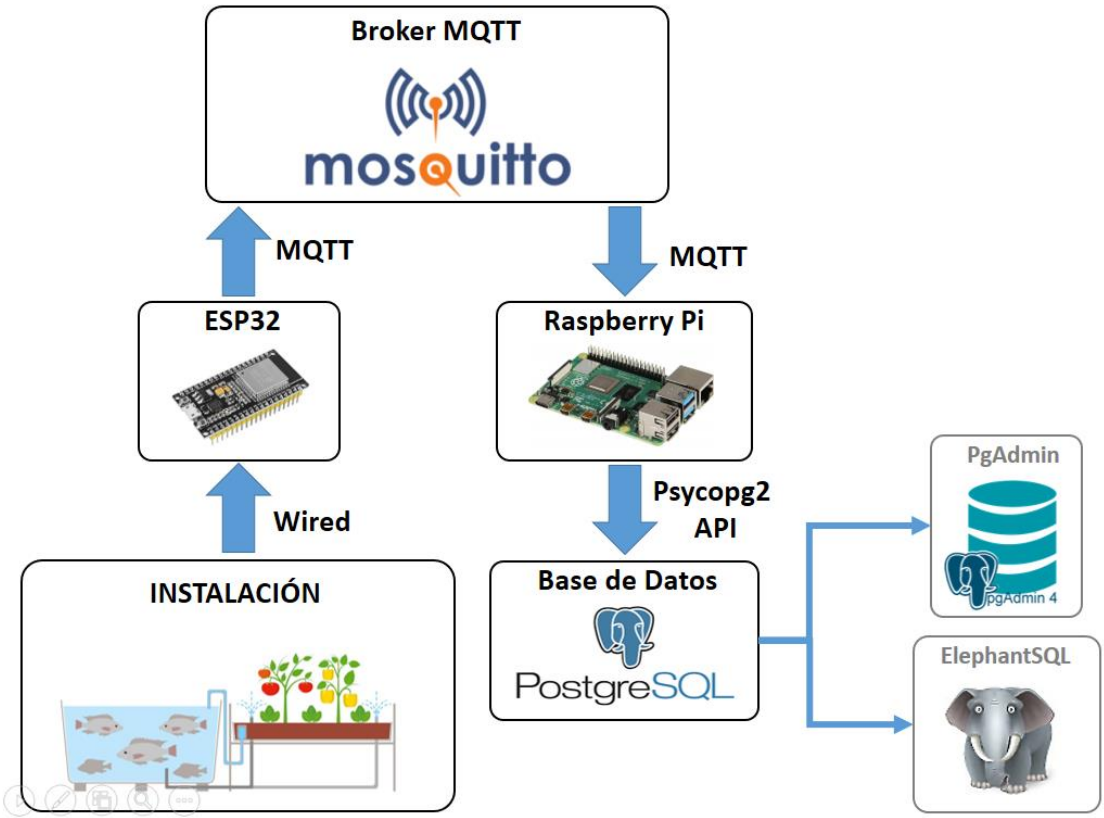


Figura 33 – Arquitectura general del sistema

3.2. Sensores

Se describe de manera detallada cada uno de los sensores integrados en la instalación.

3.2.1. DS18B20

Para medir la temperatura del agua en los tanques se incluyeron dos sensores DS18B20.

3.2.1.1. Descripción general

Sensor de temperatura digital que incorpora un elemento termosensible para medir mediante la variación de su resistencia eléctrica. El dispositivo incorpora un convertidor analógico digital (ADC) internamente.

Se utilizó el DS18B20 con encapsulado TO-92 que además viene en forma de sonda impermeable.

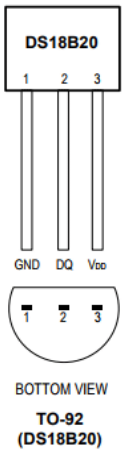


Figura 34 – Sensor DS18B20 con encapsulado TO-92

Figura 35 – Sensor DS18B20 con sonda impermeable

Utiliza el protocolo de comunicación serie 1-Wire que permite la transmisión de datos a través de un único cable. Cada dispositivo 1-Wire tiene una dirección única de 64 bits permitiendo la conexión de varios dispositivos mediante una sola línea de comunicación.

Para programarlo en ArduinoIDE serán necesarias las siguientes librerías:

- OneWire
- DallasTemperature

Se eligió este dispositivo debido a su bajo costo y consumo, altas prestaciones y su cómoda integración.

3.2.1.2. Especificaciones técnicas

- Rango de medición: -55 a 125 ºC
- Precisión: +/- 0.5 ºC (-10 a 85 ºC)
- Interfaz: 1-Wire
- Tiempo de captura: <750ms
- Alimentación: 3 a 5.5V

3.2.2. DFRobot SEN0169

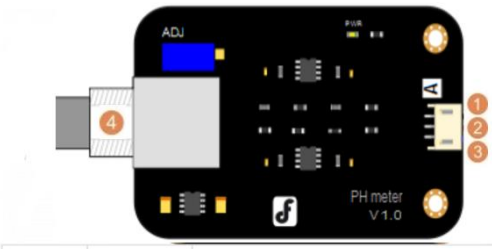
Para conocer el nivel de pH del agua se utilizó la sonda SEN0169 de DFRobot.

3.2.2.1. Descripción general

Sensor analógico de pH diseñado para medir la acidez o alcalinidad de una solución utilizando un electrodo de vidrio en una sonda con conector BNC.

Figura 36 – Sensor de Ph de electrodo de vidrío SEN0169 DFRobot

El kit contiene un modulo electrónico encargado de amplificar y procesar la señal. Este modulo presenta como entrada un conector BNC hembra, ofreciendo una interfaz de salida a tres pines (VCC, GND y ANALOG). Además incorpora un potenciómetro multivuelta para ajustar la ganancia.



No.	Label	Description
1	A	Analog Signal Output (0~3.0V)
2	+	VCC (3.3~5.5V)
3	-	GND
4	BNC	Probe Cable Connector

Figura 37 – Puertos del módulo de acondicionamiento Ph meter V1.0

3.2.2.2. Especificaciones técnicas

- Alimentación del módulo de acondicionamiento: 5,00 V
- Tamaño del módulo de acondicionamiento: 43 mm x 32 mm
- Rango de medición: 0-14PH
- Temperatura de medición: 0-60 ℃
- Precisión: ± 0,1pH (25 ℃)
- Tiempo de respuesta: ≤ 1min
- Conector de sonda: BNC

A continuación se muestra la relación entre pH y voltaje de salida que ofrece la sonda según el fabricante:

VOLTAGE （mV）	pH value	VOLTAGE （mV）	pH value
414.12	0.00	-414.12	14.00
354.96	1.00	-354.96	13.00
295.80	2.00	-295.80	12.00
236.64	3.00	-236.64	11.00
177.48	4.00	-177.48	10.00
118.32	5.00	-118.32	9.00
59.16	6.00	-59.16	8.00
0.00	7.00	0.00	7.00

Figura 38 - Relación mV-pH del SEN0169 (DFRobot)

3.2.2.3. Calibración

Los sensores analógicos pueden sufrir variaciones y desviaciones con el tiempo y uso, provocando mediciones inexactas. Por este motivo es de crucial importancia realizar un proceso de calibración a los sensores periódicamente, ajustando sus parámetros para conseguir que sus medidas se ajusten con precisión a la magnitud fisica que están midiendo.

Para calibrar ésta sonda se deben ajustar dos parámetros:

1. Offset

Este parámetro se refiere al voltaje medido cuando no hay diferencia de pH entre la solución de referencia y la solución que se está midiendo. El offset se ajusta para un valor de pH igual a 7, o lo que es lo mismo, cortocircuitando el conector BNC de entrada del modulo de acondicionamiento.

Bajo estas condiciones se obtiene el valor de pH medido calculando el offset sufrido por el sensor de la siguiente manera.

$$pH_{offset} = pH_7 - pH_{medido}$$

Dicho valor será tenido en cuenta en las mediciones.

2. Ganancia

Una vez conocido el offset se debe ajustar la pendiente de la curva pH-mV. Para ello hay que sumergir la sonda en una solución cuyo pH sea conocido. Se debe ajustar la ganancia del sensor mediante el poteciómetro que incorpora el modulo de acondicionamiento hasta que el pH medido sea el correcto.

3.2.3. DFRobot SEN0237

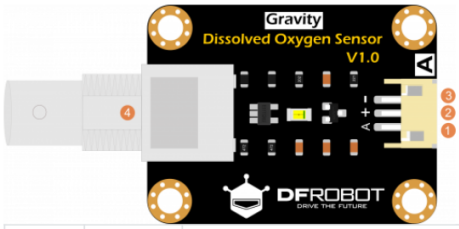
Para medir la concentración de oxígeno disuelto del agua se incorporó una sonda SEN0237 de DFRobot.

3.2.3.1. Descripción general

Sensor analógico utilizado para medir la concentración de oxígeno disuelto en una solución. Utiliza tecnología electroquímica que se basa en medir la corriente eléctrica generada por la reacción química entre el oxígeno de la solución y el electrodo de platino alojado dentro del sensor.

Figura 39 – Sensor electroquímico de oxígeno disuelto SEN0237 DFRobot

El kit contiene un modulo electrónico para acondicionar la señal. Su entrada incorpora un conector BNC hembra, ofreciendo como salida una interfaz a tres pines (VCC, GND y ANALOG).



No.	Label	Description
1	A	Analog Signal Output (0~3.0V)
2	+	VCC (3.3~5.5V)
3	-	GND
4	BNC	Probe Cable Connector

Figura 40 - Puertos módulo de acondicionamiento del SEN0237 DFRobot

3.2.3.2. Especificaciones técnicas

- **Sonda de oxígeno disuelto**
  - Tipo: sonda galvánica
  - Rango de detección: 0~20 mg/L
  - Rango de temperatura: 0~40 ℃
  - Tiempo de respuesta: hasta un 98 % de respuesta completa, en 90 segundos (25 ℃)
  - Rango de presión: 0~50 PSI
  - Vida útil del electrodo: 1 año (uso normal)
  - Longitud del cable: 2 metros
  - Conector de sonda: BNC
- **Módulo de acondicionamiento de señal**
  - Voltaje de suministro: 3,3 ~ 5,5 V
  - Señal de salida: 0 ~ 3,0 V
  - Conector de entrada: BNC
  - Conector de salida: interfaz analógica de gravedad (PH2.0-3P)
  - Dimensión: 42 mm * 32 mm/1,65 * 1,26 pulgadas

3.2.3.3. Calibración

La temperatura tiene una gran influencia sobre la concentración de oxígeno disuelto en una solución. Para una temperatura fija, el voltaje que devuelve la sonda está directamente relacionado con la concentración de oxígeno disuelto.

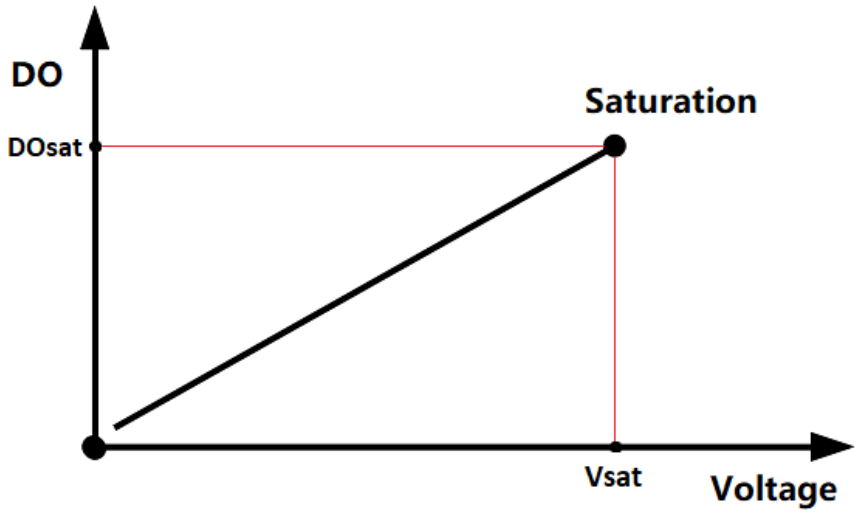


Figura 41 – Relación entre el voltaje y la concentración de oxígeno disueto a temperatura constante

En este caso, el proceso de calibración se basa en calcular el voltaje de saturación del oxígeno, que depende de la temperatura de la solución.

Para mejorar la precisión de las medidas hay que considerar los cambios en la concentración de oxígeno disuelto y en el voltaje de saturación causados por las variaciones de temperatura.

La relación entre en voltaje de saturación y la temperatura se describe mediante una curva de compensación lineal. Para calcularla es necesario conocer el voltaje de oxígeno saturado a dos temperaturas diferentes.

Para realizar una correcta calibración a dos puntos se debe seguir cuidadosamente la siguiente metodología:

1. Obtener dos vasos de agua desionizada a distinta temperatura, sin sobrepasar los 40 grados.

2. Saturar el oxígeno de ambas soluciones mediante agitación o utilizando una bomba de aire.

3. Una vez saturadas las soluciones y desaparecidas todas las burbujas de aire, sumergir la sonda en una de ellas.

4. Registrar los valotes de temperatura y voltaje una vez que este último se estabilice (Tcal_1, Vcal_1).

5. Repetir el paso 3 y 4 con la segunda solución. (Tcal_2, Vcal_2).

Los dos puntos de operación calculados definen la curva de compensación de temperatura de la sonda. De esta manera es posible calcular el voltaje de saturación de oxígeno de una solución a temperatura conocida (T) mediante la ecuación de una recta:

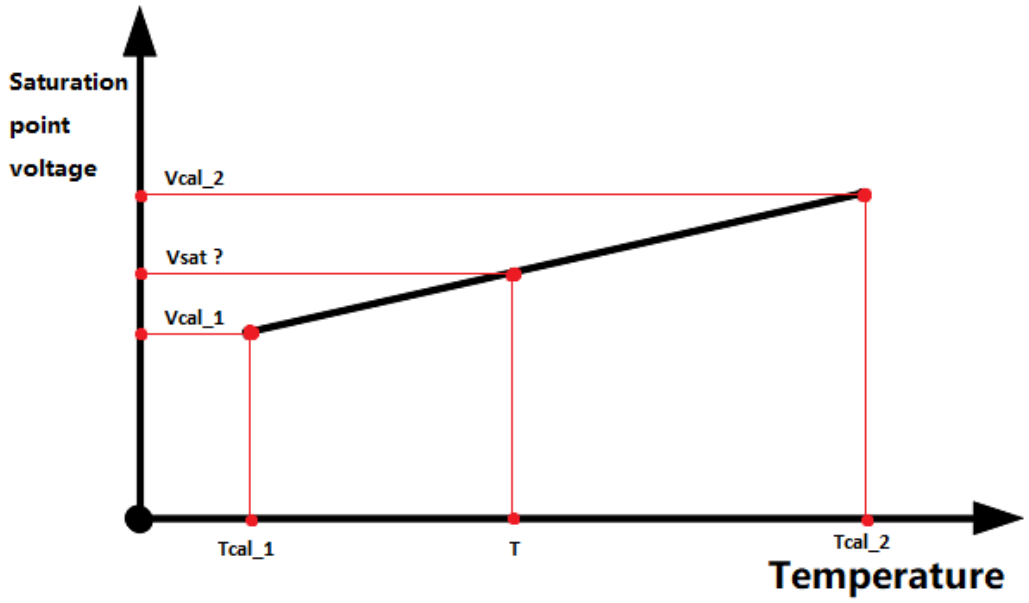


Figura 42 – Curva de compensación de temperatura

$$\frac{Vcal_2 - Vsat}{Tcal_2 - T} = \frac{Vcal_2 - Vcal_1}{Tcal_2 - Tcal_1}$$

$$\boldsymbol{Vsat = \frac{(T - Tcal_2)(Vcal_1 - Vcal_2)}{Tcal_1 - Tcal_2} + Vcal_2}$$

3.2.4. DFR0300

Para conocer la conductividad eléctrica del agua y así determinar la concentración de sales disueltas se utilizó una sonda DFR300 de DFRobot.

3.2.4.1. Descripción general

Sensor analógico de conductividad eléctrica utilizado para conocer la concentración de sales en una solución. Su principio de funcionamiento consiste en dos electrodos que se sumergen en la solución aplicando una corriente eléctrica a través de ellos. La conductividad se obtiene a partir del valor de resistencia calculado entre ambos electrodos.

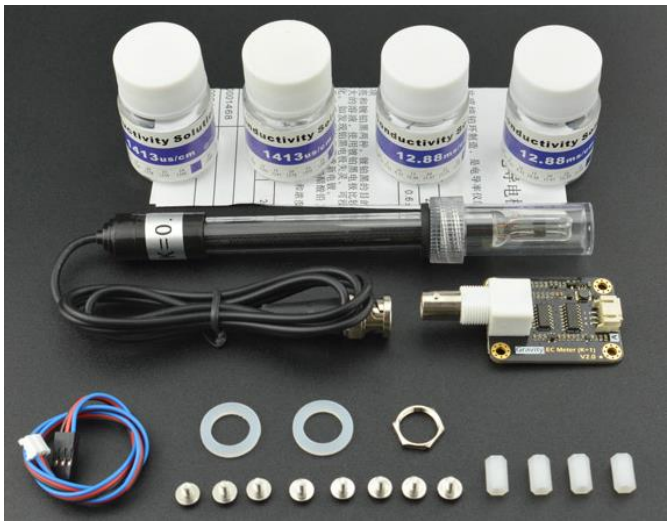
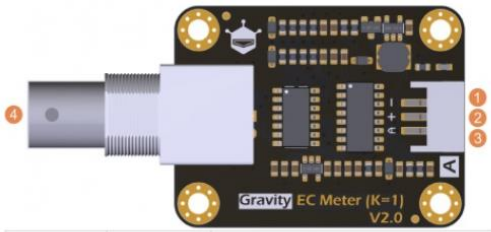


Figura 43 - Sensor de conductividad eléctrica por contacto SEN300 DFRobot (Kit completo)

Se incluye un módulo de acondicionamiento encargado de ampificar y procesar la señal. Al igual que los anteriores sensores analógicos, este módulo electrónico presenta un conector BNC hembra como entrada y posee una salida a tres pines (VCC, GND y ANALOG).



Num	Label	Description
1	-	Power GND(0V)
2	+	Power VCC(3.0~5.0V)
3	A	Analog Signal Output(0~3.4V)
4	BNC	Probe Connector

Figura 44 - Puertos módulo de acondicionamiento del SEN300 DFRobot

3.2.4.2. Especificaciones técnicas

- **Placa de conversión de señal (transmisor) V2**
  - Voltaje de suministro: 3,0 ~ 5,0 V
  - Voltaje de salida: 0 ~ 3,4 V
  - Conector de sonda: BNC
  - Conector de señal: PH2.0-3Pin
  - Precisión de medición: ±5% FS
  - Tamaño de la placa: 42 mm x 32 mm/1,65 pulgadas x 1,26 pulgadas.
- **Sonda de conductividad eléctrica**
  - Tipo de sonda: Grado de laboratorio
  - Constante de celda: 1.0
  - Rango de detección de soporte: 0~20ms/cm
  - Rango de detección recomendado: 1~15ms/cm
  - Rango de temperatura: 0~40°C
  - Vida útil de la sonda: >0,5 años (dependiendo de la frecuencia de uso)
  - Longitud del cable: 100 cm

3.2.4.3. Calibración

El valor de conductividad eléctrica medida en una solución depende de la temperatura a la que se encuentre. Por ese motivo, la calibración de esta sonda también incluye el cálculo de una curva de compensación de temperatura que deberá ser tenida en cuenta en cada medición.

Para este sensor existe una librería de ArduinoIDE específica para microncontroladores ESP32 y ESP8266 encargada de:

- Calibrar el sensor ajustando la constante de conductividad eléctrica (K).
- Calcular las medidas de conductividad teniendo en cuenta la compensación de temperatura.

Librerías necesarias:

- DFRobot_ESP_EC_BY_GREENPONIK

  Encargada de calibrar el sensor y calcular el valor de conductvidad en base al voltaje medido y la temperatura.

- EEPROM

  Necesaria para almacenar en la memoria EEPROM del microcontrolador el valor de la constante de coductividad eléctrica obtenida durante la calibración, evitando que se pierda dicha información al apagar el microcontrolador.

3.2.5. Interruptor flotador RSF88

El nivel de agua del tanque será controlado mediante el uso de dos sensores tipo float switch RSF88 colocados a diferentes alturas.

3.2.5.1. Descripción general

Un interruptor de flotador (float switch) es un tipo de sensor utilizado para detectar la presencia o ausencia de líquidos en un recipiente. Se compone de un flotador y un interruptor que se activa o desactiva cuando el nivel de líquido lo alcanza.

Es común utilizar varios sensores de este tipo para controlar el nivel de un líquido en un tanque o depósito.

Figura 45 – Detector de nivel tipo flotador (float switch)

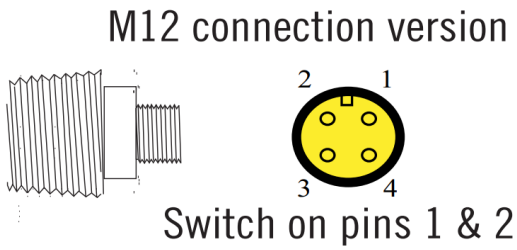


Figura 46 – Pines del sensor float switch

El dispositivo posee cuatro pines de salida, aunque solo se utilizan dos. El par de pines a utilizar dependerá del tipo de contacto que se necesite (NA / NC).

3.2.5.2. Especificaciones técnicas

- Material: Polipropileno
- Colour: Blanco
- Rango de temperatura: -20 a 100 ºC
- Viscosidad mínima del fluido (SG): 0.80
- Tipo de contacto: NA / NC
- Tensión máxima AC: 100VAC
- Tensión máxima DC: 300VDC
- Corriente máxima: 1A
- Interfaz: Conexión versión M12

3.2.6. DHT22

Los valores de temperatura y humedad relativa del ambiente serán adquiridos mediante un sensor DHT22.

3.2.6.1. Descripción general

Sensor digital de temperatura y humedad relativa. Es una versión mejorada del popular DHT11. Internamente incorpora un sensor capacitivo para la medida de humedad y un termistor para la medida de temperatura.

Utiliza un protocolo de comunicación propio que permite el envío de datos a través de un único pin de salida digital.

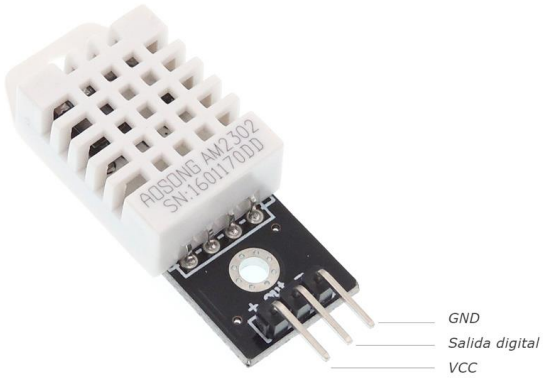


Figura 47 – Terminales del sensor ambiental DHT22

Para programar el sensor en ArduinoIDE se utiliza:

- DHT sensor library

3.2.6.2. Especificaciones técnicas

- Voltaje de alimentación: 5VDC
- Voltaje de salida: 0 – 3.3VDC
- Rango de temperatura: -40 a 80 ºC
- Precisión de temperatura: +/- 0.5 ºC
- Resolución de temperatura: 0.1 ºC (16 bits)
- Rango de humedad relativa: 0 a 100%
- Precisión de humedad relativa: +/- 2%
- Resolución de humedad relativa: 0.1% HR (16 bits)

3.3. Actuadores

Este proyecto permite la integración de dos dispositivos de actuación que no se han incluido físicamente:

- Bomba peristáltica
- Electroválvula

Para poder controlar dichos dispositivos se incluye un módulo relé de dos canales que será descrito más delante.

3.4. Microcontroladores

Se describen los microcontroladores utilizados en el desarrollo del proyecto, los cuales realizarán tareas de adquisición, procesado, envío y almacenamiento de datos.

3.4.1. ESP32 FireBeetle Board

Como elemento de adquisición de datos de la instalación y procesado se utilizó un microcontrolador FireBeetle ESP32.

3.4.1.1. Descripción general

Como dispositivo de adquisición de datos y control se utiliza un microcontrolador FireBeetle Board-ESP32 V4.0 de DFRobot. Se trata de una placa de desarrollo compacta de 34 pines basada en el microcontrolador ESP32 de Espressif Systems, un chip de bajo consumo que combina Wi-Fi y Bluetooth en una sola solución, lo que la hace ideal para aplicaciones de IoT y electrónica de consumo.

A continuación se muestra el GPIO de la placa:

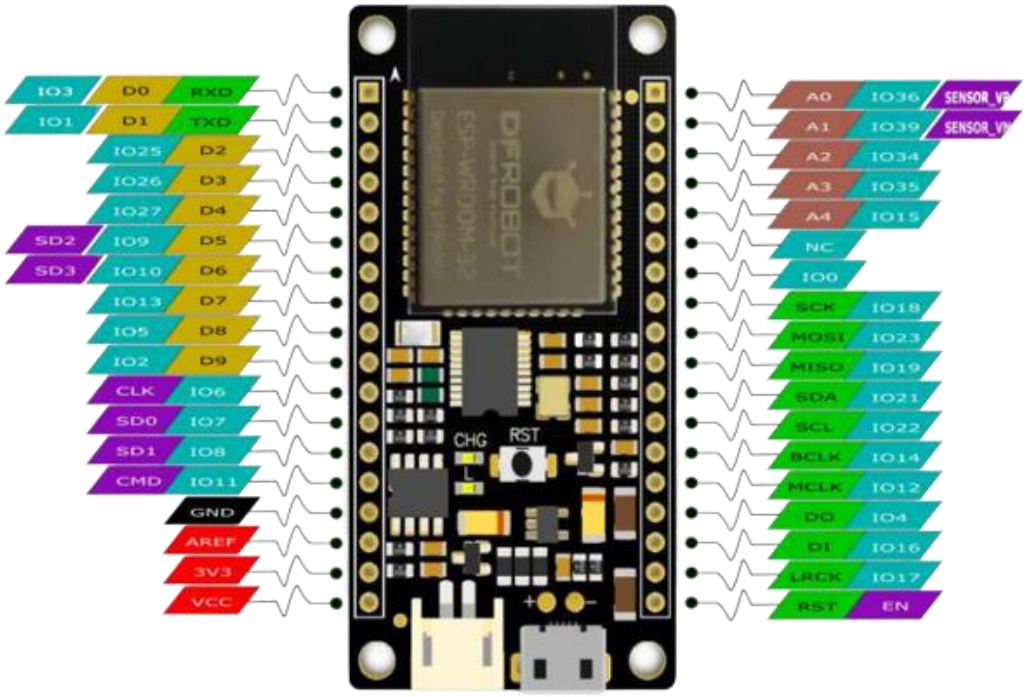


Figura 48 – Distribución de pines FireBeetle ESP32-Board

3.4.1.2. Especificaciones técnicas

- Voltaje de operación: 3.3VDC
- Voltaje de alimentación: 3.3-5VDC
- Consumo de corriente en bajo consumo: 10uA
- Corriente máxima de descarga: 600mA
- Corriente máxima de carga: 500Ma

- Procesador: Tensilica LX6 Dual Core
- Frecuencia: 240MHz
- SRAM: 520KB
- Flash: 16MB
- Estándar Wi-Fi 802.11 b
- Rango de frecuencia Wi-Fi: 2.4 - 2.5GHz
- Estándar Bluetooth v4.2
- Reloj interno: cristal de 40MHz y 32.768KHz
- Entradas/Salidas digitales: 10
- Entradas analógicas: 5
- Interfaces de comunicación: UART, SPI, I2C, I2S, ASC, DAC, 1-Wire
- Temperatura de operación: -40 a 85 ºC

3.4.2. Raspberry Pi 4B

Para realizar la conexion con la base de datos almacenando toda la información del sistema se utilizo una Raspberry Pi 4B.

3.4.2.1. Descripción general

Ordenador de placa reducida (SBC) y bajo coste desarrollado en el Reino Unido por la Raspberry Pi Foundation.

El software es open-source, siendo su sistema operativo oficial una versión adaptada de la distribución Debian (basada en Linux) denominada RaspberryPi OS (antiguamente Raspbian), aunque permite usar otros sistemas operativos.

A continuación se muestra la distribución de sus pines:

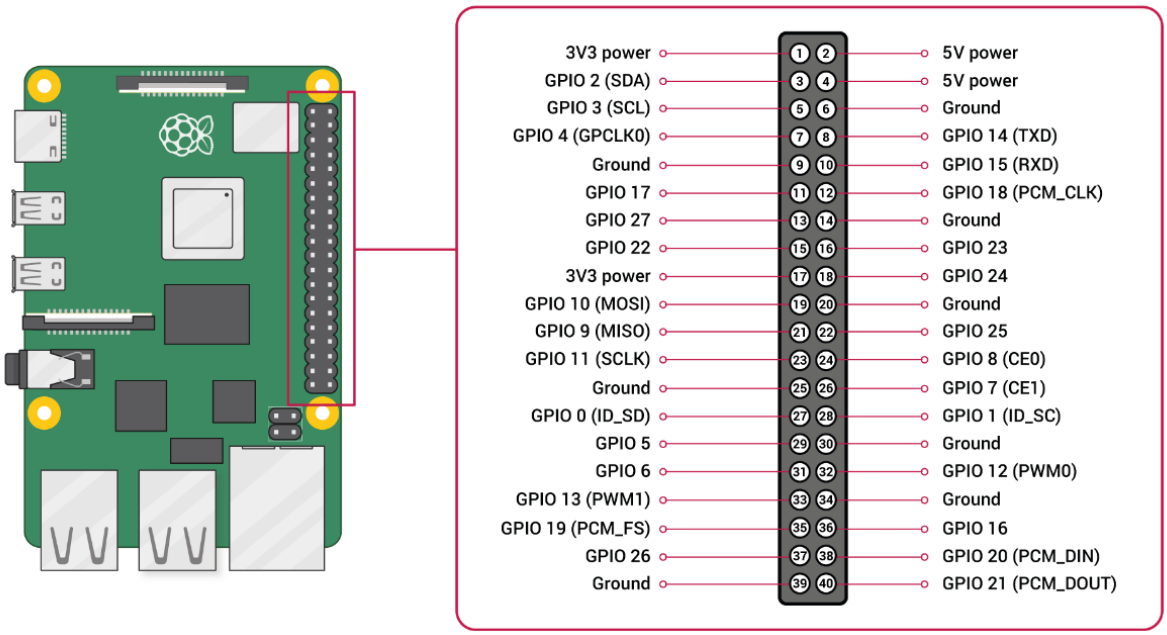


Figura 49 - Distribución de pines Raspberry Pi 4B

3.4.2.2. Especificaciones técnicas

- Procesador: Broadcom BCM2711, ARM Cortex-A72 de 64 bits (4 núcleos a 1.5 GHz)
- Memoria RAM: 8GB LPDDR4-3200 SDRAM
- Conectividad inalámbrica:
    - Wi-Fi 802.11ac de doble banda (2.4GHz y 5GHz)
    - Bluetooth 5.0
- Ethernet: Gigabit Ethernet
- Puertos USB:
    - 2 x USB 2.0
    - 2 x USB 3.0
- Puertos de video: 2 x micro-HDMI
- Audio: Conector de audio de 3.5mm para auriculares y micrófono.
- GPIO: Conector de 40 pines
- Interfaces de comunicación: UART, SPI, I2C
- Almacenamiento: Ranura para tarjeta microSD (SO y almacenamiento de datos)
- Interfaz de cámara: Conector de cámara MIPI de 2 canales
- Interfaz de pantalla: Interfaz de pantalla DSI de 2 carriles
- Dimensiones: 88 x 58 x 19.5 mm

3.5. Otros dispositivos

Además de los elementos descritos anteriormente se hace necesario incluir varios dispositos adicionales que permitan el almacenamiento de datos en local, la temporización en tiempo real, la visualización de los datos en la instalación y la conmutación de los actuadores.

3.5.1. DFR0229 (Módulo MicroSD)

Encargado de almacenar todas las medidas del sistema de manera local en una tarjeta MicroSD.

3.5.1.1. Descripción general

Módulo que permite la conexión de una tarjeta MicroSD a un microcontrolador mediante una interfaz SPI.

Cuenta con un conector para la tarjeta MicroSD, un regulador de voltaje integrado para el suministro de tensión estable y constante a la tarjeta, y un convertidor de nivel lógico de 3.3V a 5V para adaptar la señal de la interfaz SPI a diferentes niveles de tensión de entrada.

Es compatible con una amplia variedad de tarjetas MicroSD, incluyendo las tarjetas de capacidad estándar y de alta capacidad (SDHC y SDXC). Permite velocidades de transferencia de datos de alta velocidad, hasta 50 Mbps.

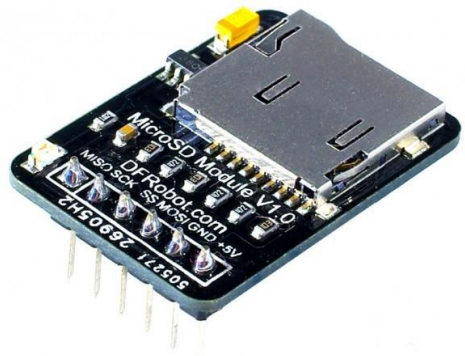


Figura 50 – MicroSD Module V1.0 DFRobot (DFR0229)

PINOUT	
MISO	Pin de comunicación de esclavo a maestro (Master Input Slave Output)
MOSI	Pin de comunicación de maestro a esclavo (Master Output Slave Input)
SCK	Reloj
SS	Pin de selección de esclavo (Slave Select)
VCC	5 VDC (Alimentación)
GND	0 VDC (Tierra)

Tabla 5 – Distribución de pines del módulo MicroSD DFR0229

Para trabajar con él en el entorno de Arduino es necesario instalar las siguientes librerías:

- SD (Secure Digital)

  Librería software de ArduinoIDE que permite leer y escribir datos en una tarjeta SD o MicroSD utilizando un microcontrolador. Proporciona una API para acceder al contenido de la memoria.

- FS (File System)

  Librería software de ArduinoIDE que proporciona una interfaz de programación para el acceso y manipulación de sistemas de archivos en un microcontrolador. Es una depedencia de la librería SD.

3.5.1.2. Especificaciones técnicas

- Voltaje de alimentación: 5 VDC
- Voltaje de operación: 3.3 VDC
- Interfaz de comunicación: SPI (Serial Peripheral Interface)
- Frecuencia de reloj máxima SPI: 20 MHz
- Soporte de sistema de archivos: FAT16 y FAT32
- Tamaño: 15mm x 25mm

3.5.2. RTC DS3231

Para obtener la marca de tiempo de cada medida de manera precisa se incluye un módulo de reloj en tiempo real (RTC) DS3231.

3.5.2.1. Descripción general

Módulo de reloj en tiempo real que utiliza un oscilador de cristal compensado por temperatura. Incluye una pila para alimentación propia y una memoria EEPROM AT24C32 capaz de almacenar 4KB de datos de forma permanente. Utiliza el protocolo de comunicación I2C.

Figura 51 – Reloj en tiempo real RTC DS3231

PINOUT	
SCL	Terminal para la señal de reloj (Serial Clock)
SDA	Terminal para la señal de datos (Serial Data)
VCC	5 VDC (Alimentación)
GND	0 VDC (Tierra)

Tabla 6 – Distribución de pines I2C del RTC DS3231

Para trabajar con él en el entorno de Arduino se utiliza la librería:

- RTCLib

3.5.2.2. Especificaciones técnicas

- Voltaje de alimentación: 3.3 - 5VDC
- Precisión del reloj: +/- 2ppm
- Interfaz de comunicación: I2C
- Dirección I2C:
  - Lectura: 0xD1
  - Escritura: 0xD0
- Memoria EEPROM AT24C32: 4KB
- Dimensiones: 38 x 22 mm

3.5.3. Display LCD 2004 I2C

Se integró un módulo display LCD de 20x4 caracteres con comunicación I2C para mostrar la información del sistema en tiempo real.

3.5.3.1. Descripción general

Módulo de visualización alfanumérica que cuenta con una pantalla de cristal líquido (LCD) capaz de mostrar 20 caracteres en cada una de sus 4 filas, sumando un total de 80 caracteres.

Esta version del módulo incorpora una pequeña placa electrónica a su reverso que permite la comunicación I2C, facilitando la conexión con otros dispositos y reduciendo la cantidad de pines necesarios. Contiene un potenciómetro para ajustar el contraste de la pantalla.

Figura 52 – Display LCD de 20x4 caracteres con módulo de comunicación I2C

Al implementar una comunicación I2C, el pinout es el mismo que el descrito en la tabla 6.

Para trabajar con la pantalla en ArduinoIDE se utiliza la librería:

- LiquidCrystal_I2C

3.5.3.2. Especificaciones técnicas

- Tamaño de pantalla: 20 x 4 caracteres
- Controlador: HD44780
- Interfaz de comunicación I2C
- Tamaño del módulo: 98 x 60 x 14 mm
- Voltaje de alimentación: 5 VDC
- Temperatura de operación: -20 a 70 ºC
- Contraste ajustablemediante potenciómetro
- Bajo consume energético

3.5.4. Relay Module 2CH

Para controlar la activación y desactivación de los actuadores se incorpora un módulo relé de dos canales.

3.5.4.1. Descripción general

Un relé es un interruptor electro-mecánico. En su interior aloja un electroimán que al excitarse unirá dos contactos metálicos, cerrando el circuito al que está conectado y activando la carga correspondiente. Sirven para activar circuitos que tienen un consumo considerable de electricidad mediante otro circuito de pequeña potencia.

Por seguridad existe un aislamiento galvánico entre las zonas de baja y alta potencia. Esto quiere decir que no existe un camino físico de conducción eléctrica entre ambas zonas, protegiendo así los elementos de control de cualquier fallo o sobretensión eléctrica.

Figura 53 – Módulo de relé de dos canales

PINOUT (Zona de baja potencia)	
IN1	Terminal de control del canal 1
IN2	Terminal de control del canal 2
VCC	5 VDC (Alimentación)
GND	0 VDC (Tierra)

Tabla 7 - Ditribución de pines del relé

3.5.4.2. Especificaciones técnicas

- Número de canales: 2
- Limites eléctricos en zona de alta potencia:
  - 10A, 250VAC
  - 10A, 30VDC
- Límites eléctricos en zona de baja potencia:
  - 10A, 125VAC
  - 10A, 28VDC

4 DESARROLLO

Una vez descritos todos los dispositivos necesarios para la instalación se procede a explicar el desarrollo y la metodología seguida en el proyecto. Para ello, éste punto se divide en dos partes, el desarrollo hardware y el desarrollo software, ejecutados ambos en paralelo.

4.1. Hardware

Se realizó un desarrollo hardware ordenado y escalado, dividido en tres etapas:

1. Integración en protoboard
2. Desarrollo en placa perforada
3. Diseño PCB

4.1.1. Integración en protoboard

Etapa inicial del desarrollo en la que se realiza un primer conexionado en protoboard, permitiendo familiarizarse con los dispositivos y realizar las primeras pruebas de forma rápida y flexible. Esta etapa es fundamental para verificar la funcionalidad básica del diseño y realizar pruebas iniciales sin incurrir en los costos y el tiempo asociados con el desarrollo de placas más permanentes.

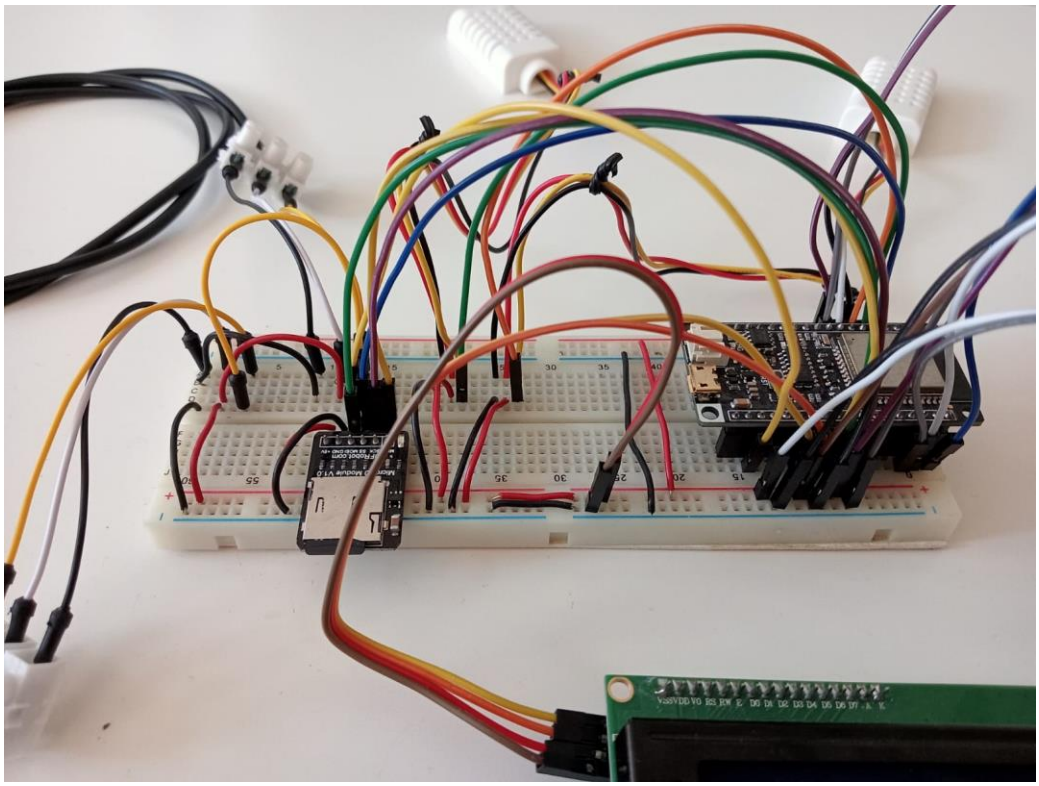


Figura 54 – Integración del sistema en protoboard (1)

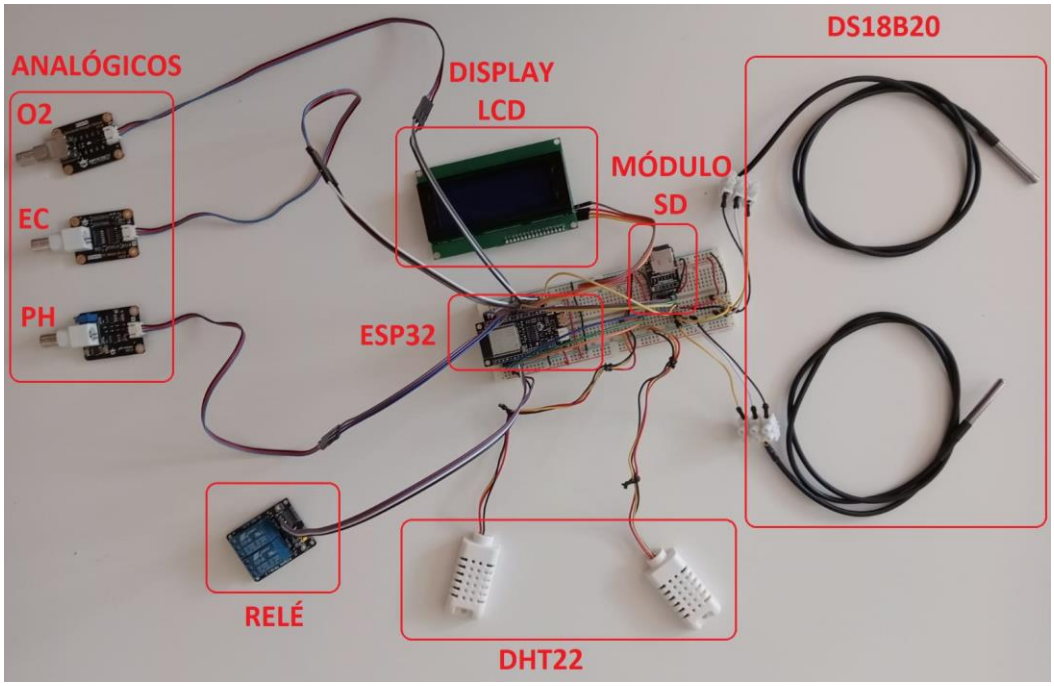


Figura 55 - Integración del sistema en protoboard (2)

4.1.2. Desarrollo en placa perforada

Una vez probado el diseño en protoboard, la siguiente etapa consiste el desarrollo en una placa perforada soldando todos los componentes. Esta etapa es una transición entre el prototipo inicial y el diseño final en una placa de circuito impreso (PCB). La placa perforada brinda una plataforma más estable y duradera para llevar a cabo pruebas más exhaustivas evaluando el rendimiento del circuito en condiciones más cercanas a la realidad.

Se incorporan dos nuevos componentes:

- Reloj de tiempo real (RTC DS3231)
- Sensores float switch

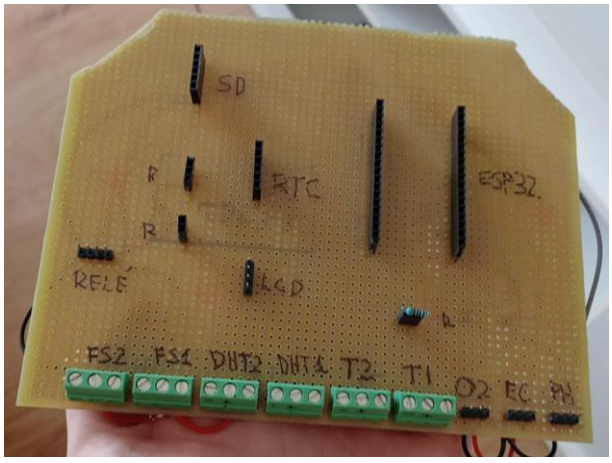


Figura 56 – Desarrollo de placa perforada (Anverso)

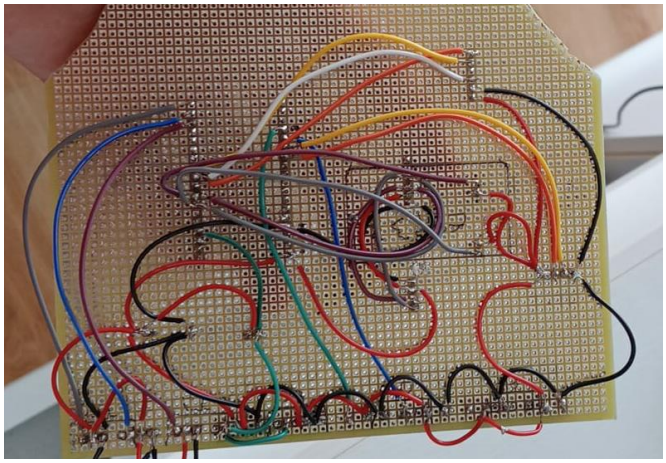


Figura 57 - Desarrollo de placa perforada (Reverso)

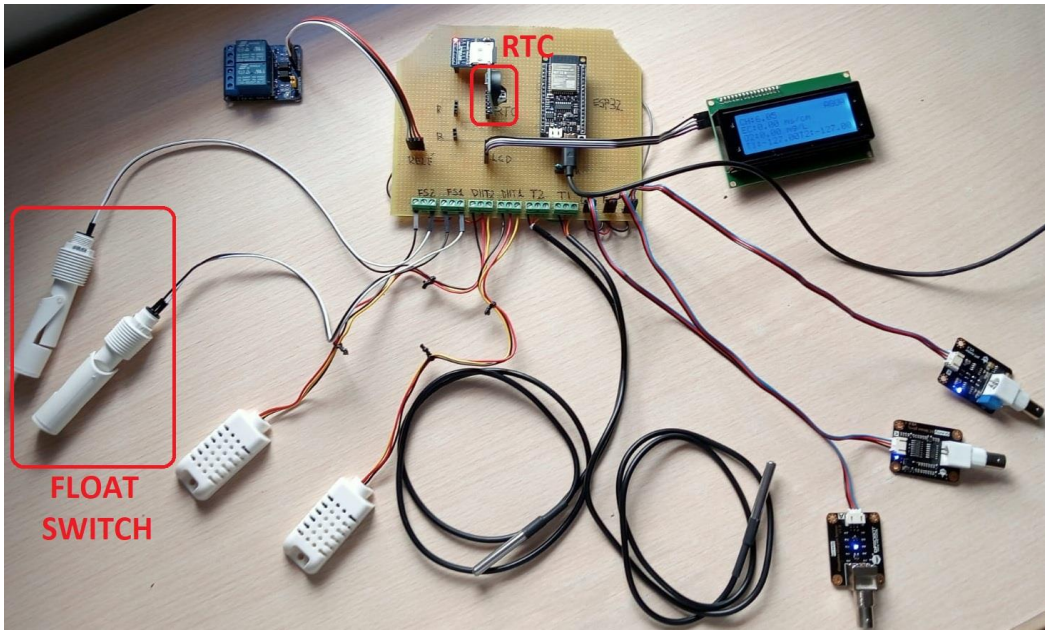


Figura 58 - Integración del sistema en placa perforada

4.1.3. Diseño PCB

La etapa final se corresponde con el diseño de la PCB. Aporta un enfoque más ordenado y compacto, mayor integridad de las señales, mejor protección contra interferencias electromagnéticas y una mayor confiabilidad con respecto a las soluciones anteriores. Además, el uso de una PCB permite la producción en masa de manera eficiente, replicando fácilmente el diseño. Para ello se utilizó el software gratuito de diseño electrónico EasyEDA.

A continuación, se muestran varias imágenes del proceso, desde el esquemático y el routing, hasta los modelos 2D y 3D de la PCB terminada. Los planos y circuitos se encuentran en el Anexo.

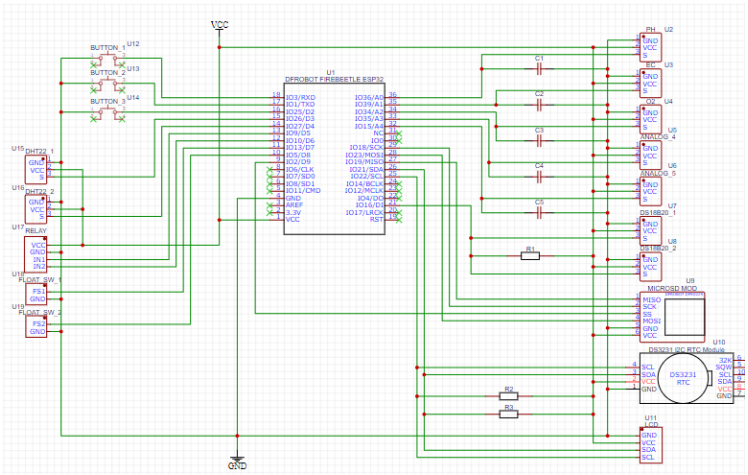


Figura 59 – Esquemático realizado con EasyEDA

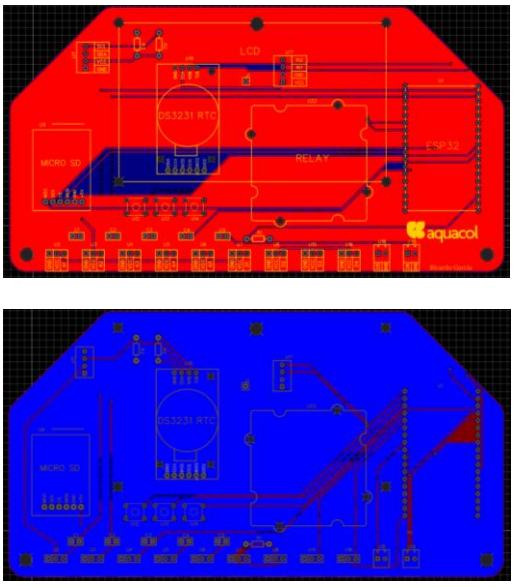


Figura 60 – Capas superior e inferior PCB en EasyEDA

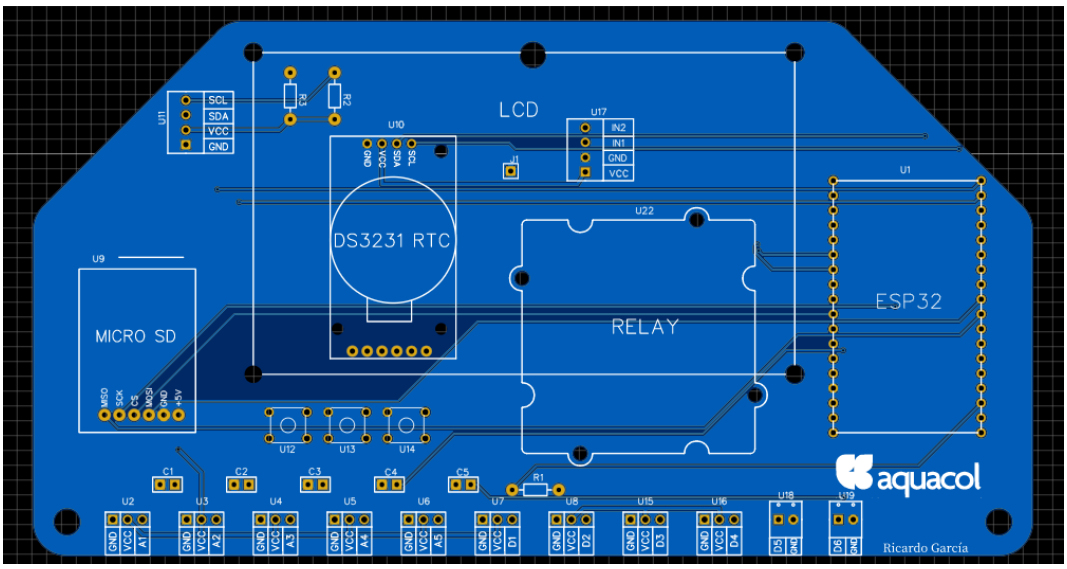


Figura 61 – Modelo 2D de la PCB en EasyEDA

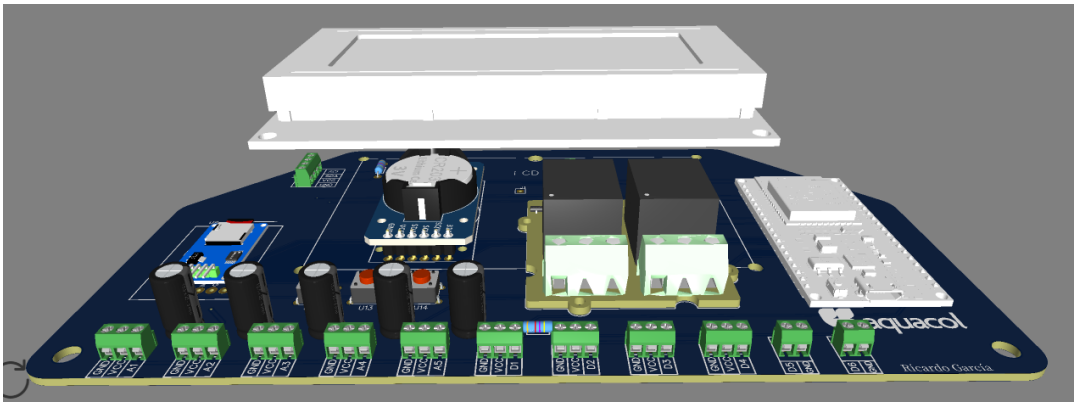


Figura 62 - Modelo 3D de la PCB en EasyEDA (1)

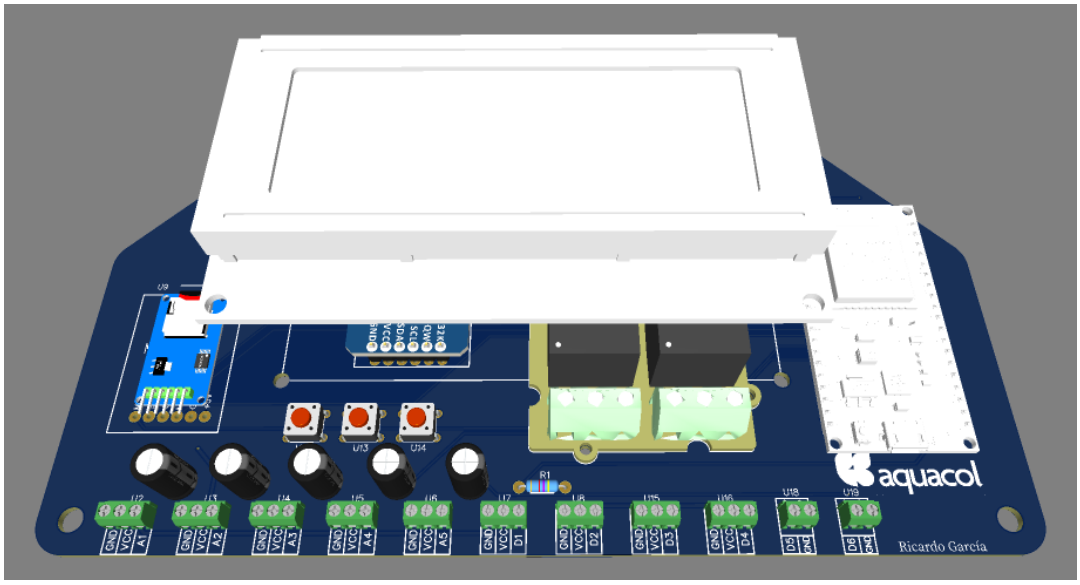


Figura 63 - Modelo 3D de la PCB en EasyEDA (2)

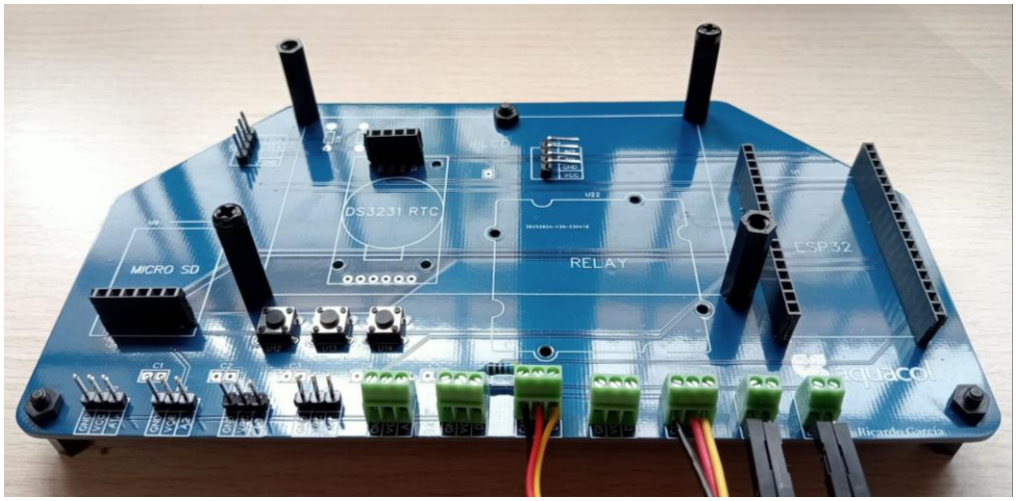


Figura 64 - PCB AQUACOL (1)

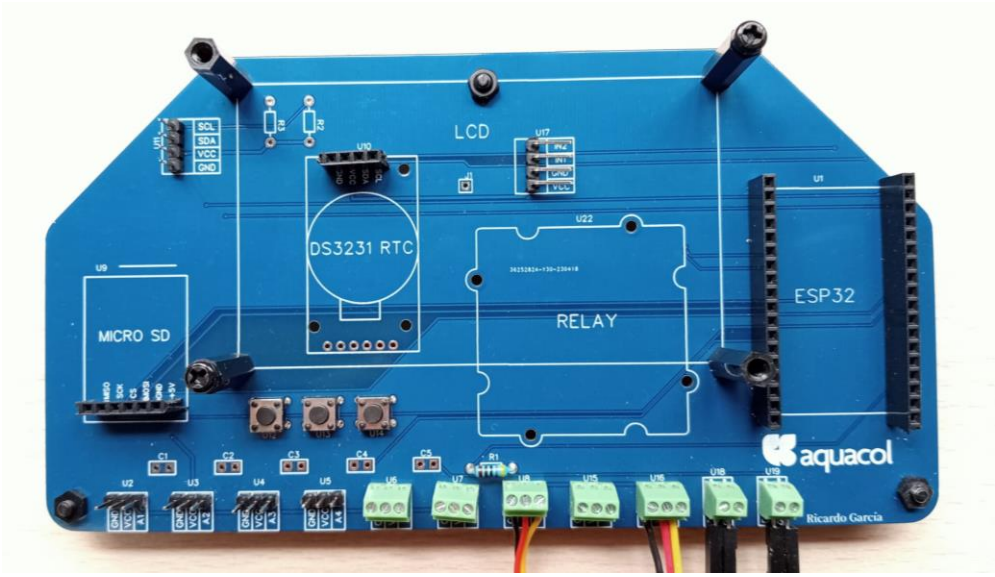


Figura 65 - PCB AQUACOL (2)

4.2. Software

El desarrollo del software se divide en dos partes:

La primera se encarga de la adquisición y procesado de datos en la instalación acuapónica, enviando dicha información por MQTT. Para ello se utiliza el microcontrolador ESP32 bajo el entorno ArduinoIDE.

La segunda parte se encarga del almacenamiento de información en una base de datos PostgreSQL en la nube. Para ello se utiliza una Raspberry Pi 4B, el broker MQTT Mosquitto y la API “psycopg2”, que permite le conexión y gestión de bases de datos Postgre desde Python.

La Raspberry Pi recibe los datos de la instalación enviados por el ESP32 mediante MQTT. Una vez recibidos y procesados son insertados en una tabla de la base de datos con una marca de tiempo.

4.2.1. Instalaciones previas

En primer lugar, se instalan los softwares, drivers y paquetes necesarios para realizar el programa de adquisición de datos en ArduinoIDE.

4.2.1.1. ArduinoIDE

Entorno de programación diseñado para trabajar con placas Arduino, aunque también es compatible con ESP32, ESP8266 o STM32 entre otras, siempre que se utilicen las librerías adecuadas.

Se instala desde su web oficial: https://www.arduino.cc/

En el siguiente enlace se muestra una guía detallada para su descarga e instalación: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing

A continuación se muestra el entorno de trabajo de ArduinoIDE

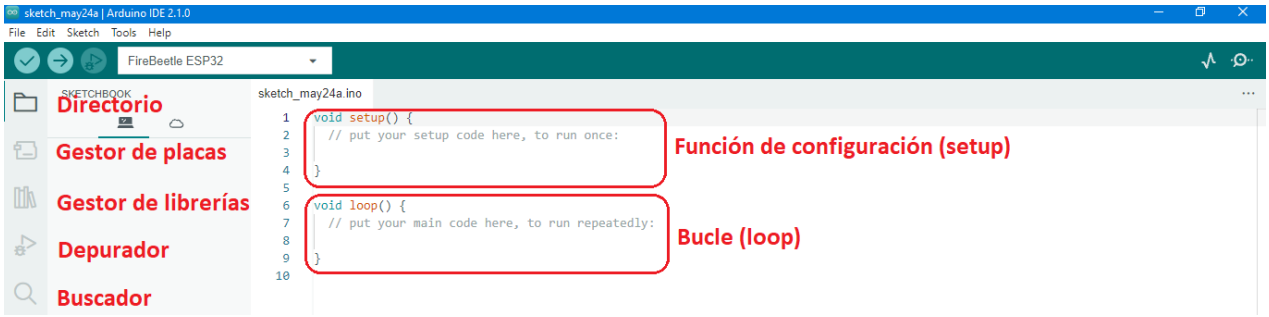


Figura 66 – Entorno ArduinoIDE

4.2.1.2. Driver ESP32

Es necesario conocer el cotrolador USB serial que posee el ESP32 para que Windows reconozca el dispositivo. Dependiendo del tipo de placa puede incorporar el CP210x o el CH340. Dicha información se puede conocer a través del data sheet de la placa de desarrollo o mirando directamente sobre la serigrafía del componente:

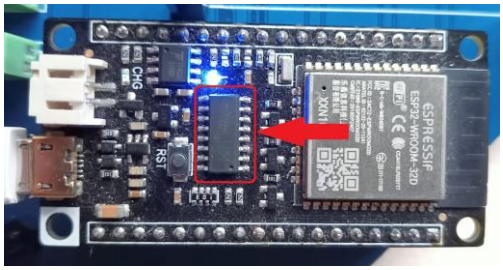


Figura 67 – Controlador CH340 del FireBeetle ESP32

El controlador CP120x se puede encontrar en la web oficial de su fabricante, SiliconLabs:
https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads

El controlador CH340 se puede instalar a través del siguiente enlace:
https://moviltronics.com/driver-ch340-placas-arduino/
El FireBeetle ESP32 utilizado en el proyecto incorpora este driver.

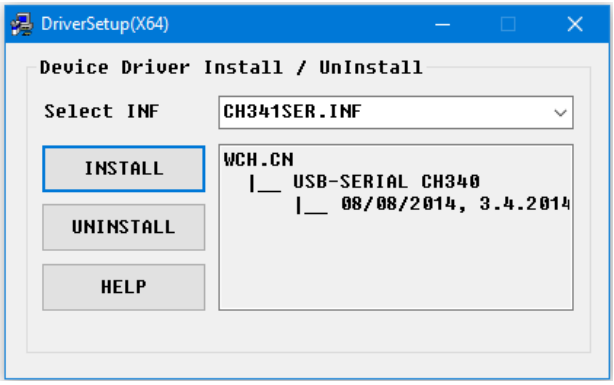


Figura 68 – Instalación del controlador CH340 (paso 1)

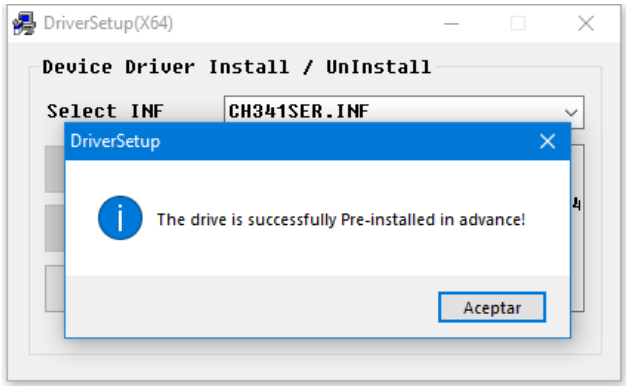


Figura 69 - Instalación del controlador CH340 (paso 2)

Al conectar el ESP32 a un puerto USB del ordenador, éste debe reconocerlo como un puerto COM, mostrando el nombre del driver. Esto se puede comprobar a través del administrador de dispositivos de Windows, en la sección de puertos COM.

4.2.1.3. Package ESP32

Para poder trabajar con el FireBeetle ESP32 en ArduinoIDE hay que instalar su correspondiente paquete. Para ello se accede a “Files > Preferences” y se incorpora el siguiente enlace que incluye todos los paquetes necesarios para trabajar con placas ESP32 de DFRobot.

http://download.dfrobot.top/FireBeetle/package_DFRobot_index.json

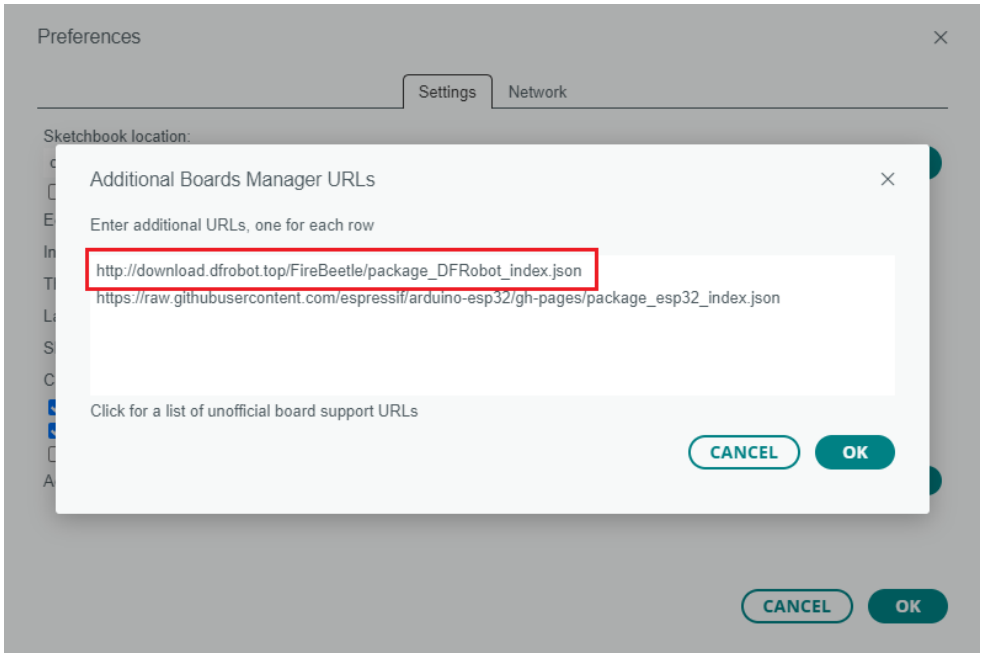


Figura 70 – Incluir URL del paquete a descargar en ArduinoIDE (package_DFRobot_Index.json)

Al aceptar se descargan automáticamente, apareciendo en el gestor de placas para proceder con su instalación, como se muestra en la siguiente figura.

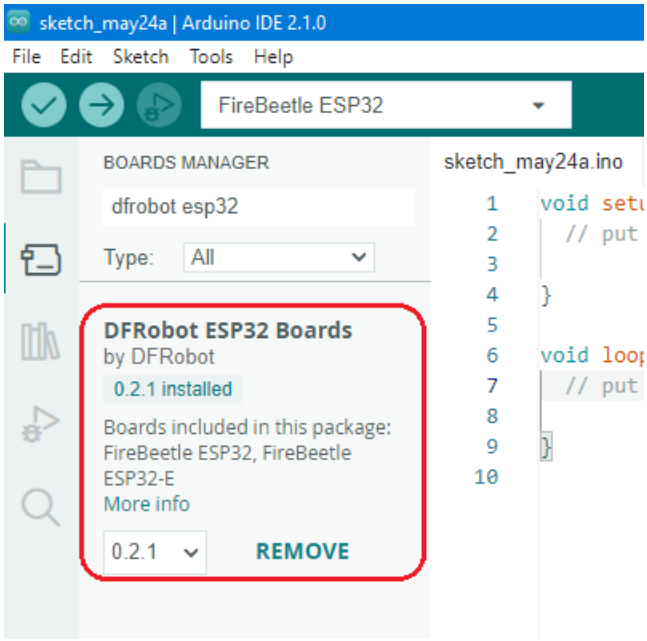


Figura 71 – Instalar paquete ESP32 desde el gestor de placas de ArduinoIDE

4.2.2. Programa de adquisición de datos (ESP32 + ArduinoIDE)

Una vez preparado el entorno de ArduinoIDE se realiza el programa de adquisición de datos.

Éste se encarga de leer y procesar la información de los sensores, almacenando dichos datos localmente en una memoria SD y enviándolos mediante MQTT al broker para su posterior almacenamiento en la base de datos.

También se realizan dos automatizaciones, una para el control de nivel de agua en un tanque y otra para el control del pH.

La información de los sensores se muestra en tiempo real a través de una pantalla LCD incorporada en la PCB.

4.2.2.1. Librerías

Las librerías son componentes de software esenciales en el entornode ArduinoIDE. Son códigos predefinidos con funciones que permiten acceder y utilizar carcaterísticas específicas del hardware, permitiendo realizar tareas complejas sin tener que escribir el código desde cero.

Una parte de las librerías que se utilizan vienen por defecto en el paquete de ESP32. Las demás se deben instalar manualmente desde el gestor de librerías del entorno. A continuación se describe cada una de ellas:

Nombre	**Descripción**
WiFi	Librería estándar incluida en el entorno que permite contectar el hardware a redes inalámbricas y utilizar protocolos de comunicación como TCP/IP para intercambio de datos con otros dispositivos de la red.
SD	Librería estándar incluida en el entorno que permite trabajar con tarjetas de memoria SD y microSD.
FS	Librería estándar incluida en el entorno que permite accede y manejar sistemas de archivos en dispositivos de almacenamiento, como tarjetas SD/microSD o memorias flash SPIFFS.
EEPROM	Librería estándar incluida en el entorno que permite utilizar la memoria EEPROM de la placa.
SPI	Librería estándar incluida en el entorno que facilita la comunicación SPI, proporcionando funciones para configurar y controlar dicha comunicación.
OneWire	Librería externa utilizada para comunicar dispositivos que siguen el protocolo OneWire, como los sensores de temperatura DS18B20, memorias RAM y otros dispositivos compatibles.
DallasTemperature	Librería externa utilizada para trabajar con sensores de temperatura basados en el protocolo OneWire, como el

	DS8B20, proporcionando una interfaz sencilla para obtener las lecturas.
DFRobot_ESP_EC_BY_GREENPONIK	Librería externa utilizada para trabajar con sensores de conductividad eléctrica de DFRobot. Proporciona una inerfaz que permite leer y calibrar el sensor cómodamente.
LiquidCrystal_I2C	Librería externa utilizada para controlar pantallas LCD mediante el protocolo I2C.
PubSubClient	Librería externa utilizada para implementar comunicaciones mediante protocolo MQTT.
DHT sensor library	Librería externa utilizada para interactuar con sensores de temperatura y humedad de la serie DHT (DHT11, DHT22).
Adafruit Unified Sensor	Librería externa desarrollada por Adafruit Industries que proporciona una interfaz unificada para trabajar con una gran variedad sensores y módulos en ArduinoIDE. Es una dependencia de la librería “DHT sensor library”.
RTClib	Librería externa utilizada para trabajar con relojes de tiempo real (RTC).
Adafruit BusIO	Librería externa desarrollada por Adafruit Industries que proporciona una interfaz común para trabajar con diversos buses de comunicación en el entorno de ArduinoIDE. Facilita la comunicación con dispositivos que utilzan I2C, SPI y UART. Es una dependencia de la librería “RTClib”.
Esp_adc_cal	Librería específica para ESP32 que proporciona funciones para calibrar y obtener lecturas precisas de su ADC.

Tabla 8 – Descripción de librerías para el código de adquisición de datos con ESP32

4.2.2.2. Descripción de funciones

El código de adquisición de datos se compone de diferentes partes:

1. Definición de constantes, variables y objetos
2. Definición de funciones
3. Configuraciones
4. Bucle de procesado.

Se trabaja con variables globales, careciendo casi todas las funciones de argumentos de entrada y salida.

A continuación, se describen las funciones implementadas.

Nombre	Descripción
Get_pH_value	Calcula y devuelve el valor de pH. Para ello obtiene ocho medidas, eliminando las dos que poseen valores extremos y calculando la media con las seis restantes. De esta forma se asegura de devolver valores coherentes y eliminar el ruido. Es necesario aplicar una función de calibración del ADC para obtener resultados más precisos.
Get_o2_value	Calcula y devuelve la concentración de oxígeno disuelto. Para ello necesita el voltaje medido, la temperatura actual y la tensión de saturación del oxígeno a dicha temperatura. Es necesario aplicar una función de calibración del ADC para obtener resultados más precisos.
Get_ec_value	Calcula y devuelve el valor de la coductividad eléctrica. Utiliza una librería específica para obtener la conductividad a partir del voltaje medido. Es necesario aplicar una función de calibración del ADC para obtener resultados más precisos.
Get_Sensors	Recopila las lecturas de todos los sensores filtrando los valores erróneos o incoherentes. Realiza la suma de cada lectura para calcular el valor medio de ellas más adelante.
Get_Average	Calcula el valor medio de todas las lecturas y obtiene la marca de tiempo.
Write_SD	Almacena el valor medio de las lecturas y la marca de tiempo en un archivo CSV alojado en la memoria SD.
Send_MQTT	Construye una trama con el valor medio de todas las lecturas y la marca de tiempo y la envía por MQTT.
Show_LCD	Muestra por la pantalla LCD el valor de cada lectura en tiempo real.
ReadFile	Lee un archivo de texto alojado en la memoria SD para obtener el SSID y la contraseña del WiFi. Se llama desde la función de configuración del WiFi (setup_wifi)
WriteFile	Genera el archivo CSV en la memoria SD donde almacenar los datos en caso de que no exista, incluyendo una cabecera. Se llama desde la función de configuración de la memoria SD (setup_sd).
AppendFile	Añade información al archivo CSV de la memoria SD. Se llama desde la función de escritura de datos en la SD (Write_SD).

Level_Control	Realiza el control de nivel de un tanque. Para ello lee el estado de los sensores float switch, activando o desactivando una electroválvula cuando corresponda. La electroválvula irá conectada al canal 1 del relé.
Ph_Control	Realiza el control del pH de un tanque. Para ello se comprueba el valor del pH cada hora. Si está por debajo del umbral se activará la bomba peristáltica durante 10 minutos. La bomba irá conectada al canal 2 del relé.
Init_times	Función encargada de inicializar las variables que alamacenan el valor de los instantes anteriores. Dichos valores son necesarias para la temporización de cada una de las tareas que se ejecutan en el bucle de procesado.
Setup_devices	Se encarga de inicializar todos los dispositivos, establecer los pines de entrada/salida y calibrar el ADC.
Setup_wifi	Función para configurar e iniciar la conexión WiFi. Obtiene el SSID y la contraseña de un archivo de texto alojado en la memoria SD. En caso de no conectarse, esperará un timeout de 15 segundos antes de notificarlo y continuar con el resto de configuraciones.
Setup_mqtt	Función para configurar e iniciar la conexión MQTT con el broker. En caso de no conectarse, esperará un timeout de 15 segundos antes de notificarlo y continuar con el resto de configuraciones.
Setup_sd	Función para configurar la memoria SD. Si en la SD no existe el archivo CSV donde alojar los datos, lo crea y le añade una cabecera.
Setup	Función de configuración típica de ArduinoIDE. Se encarga de realizar las llamadas a todas las configuraciones descritas anteriormente.
Loop	Bucle de procesamiento. Se realizan las diferentes tareas de manera cooperativa, realizando las llamadas a todas las funciones descritas anteriormente.

Tabla 9 - Descripción de funciones para el código de adquisición de datos con ESP32

4.2.2.3. Descripción del código

A continuación se describe el código de adquisición de datos mediante un diagrama de flujo:

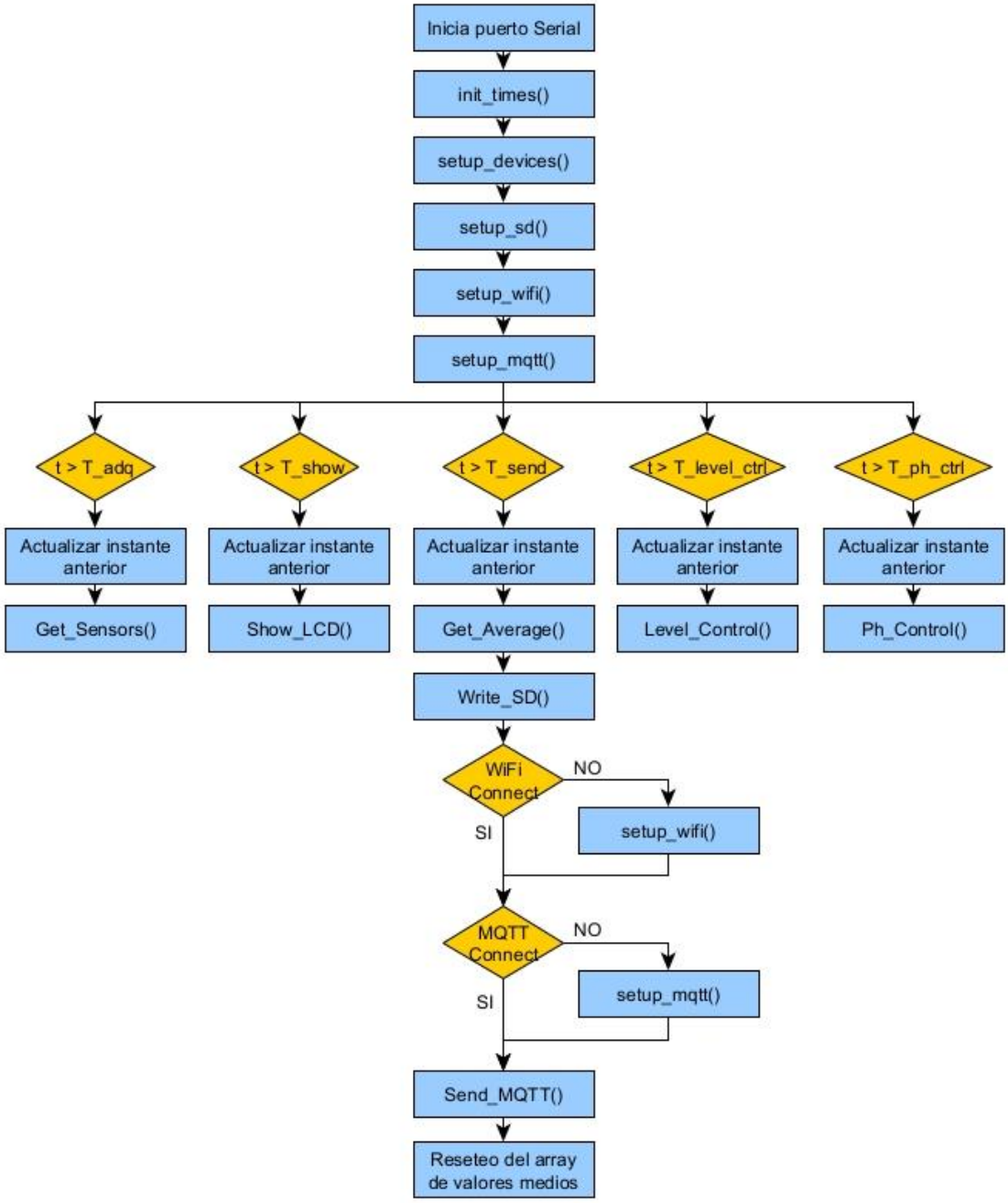


Figura 72 – Diagrama de flujo referente al código de adquisición de datos

4.2.3. Configuración Raspberry Pi

Antes de poder trabajar con la Raspberry es necesario realizar una serie de configuraciones previas, que incluyen:

1. Preparación de la tarjeta microSD
2. Escritorio remoto
3. IP estática
4. Instalación de paquetes

4.2.3.1. Preparación de la tarjeta SD

El primer paso es cargar el sistema operativo en la memoria SD que incluye la placa, estableciendo unos credenciales de acceso (nombre de usuario y contraseña) y configurando la conexión WiFi. Para ello se utiliza Raspberry Pi Imager, que se puede descargar a través del siguiente enlace:

https://www.raspberrypi.com/software/

Se describe el proceso a seguir para la configuración de la microSD:

1. Contar con una tarjeta microSD de 8GB como mínimo y formatearla. Para ello se utilizó el software “SD Card Formatter”.

2. Cargar el SO en la memoria microSD, configurando los credenciales de acceso, la conexión WiFi y el servidor SSH con Raspberry Pi Imager.

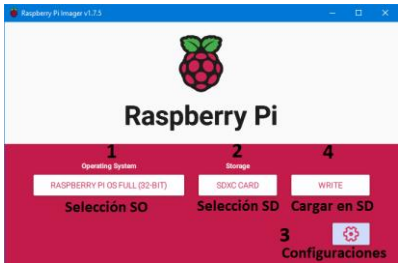


Figura 73 – Pasos para configurar y cargar Raspberry Pi OS en MicroSD con RaspberryPi Imager

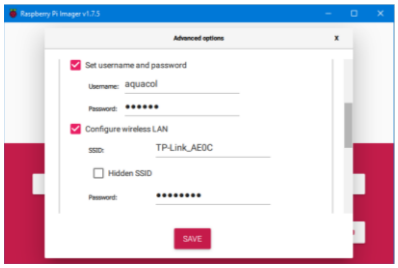


Figura 74 – Configuración de los credenciales y el WiFi

Figura 75 – Cargando SO en MicroSD

3. Una vez cargado el SO, hay que extraer y volver a insertar la tarjeta en el PC. El contenido de la microSD queda de la siguiente manera:

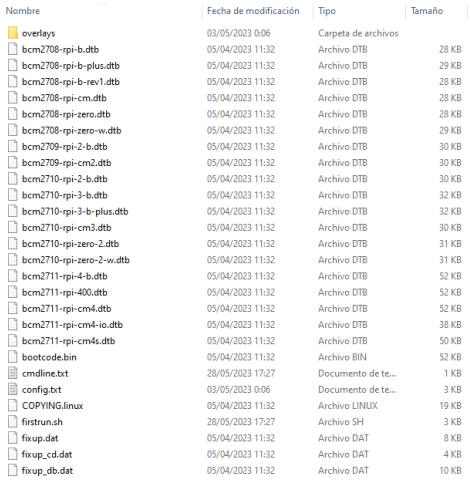


Figura 76 – Contenido de la MicroSD con el SO

4.2.3.2. Escritorio remoto

Como todo ordenador, la Raspberry Pi incorpora los puertos necesarios para trabajar con ella utilizando ratón, teclado y monitor. En este caso se prescinde de dichos periféricos utilizando un escritorio remoto llamado VNC Viewer, disponible en:

https://www.realvnc.com/es/connect/download/viewer/

Para acceder a la Raspberry se necesita conocer la dirección IP de la placa dentro de la red local. Para ello se puede utilizar algún software de escaneo de IPs, como “Advanced IP Scanner”.

Es necesario habilitar el escritorio remoto VNC en la Raspberry. Para ello se accede al sistema mediante un servidor SSH, utilizando la dirección IP asignada a la placa. Se utlizó el software “Putty”.

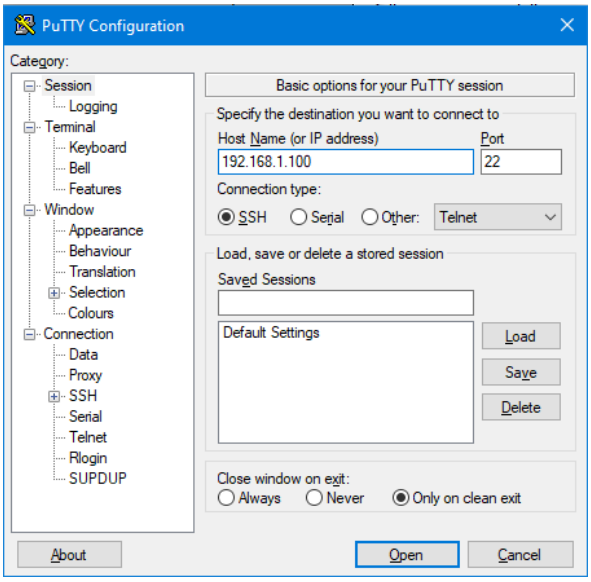


Figura 77 – Conexión remota con la Raspberry utilizando servidor SSH en Putty

Una vez dentro, se utiliza la herramienta de configuración del sistema (raspi-config) para habilitar el servicio VNC.

Interface Options > VNC > Enable

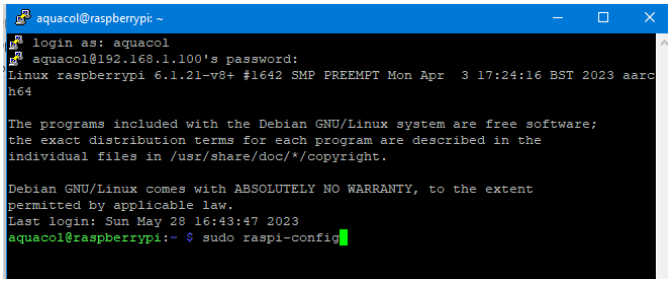


Figura 78 – Comando para abrir la herramienta de configuración de Raspberry Pi

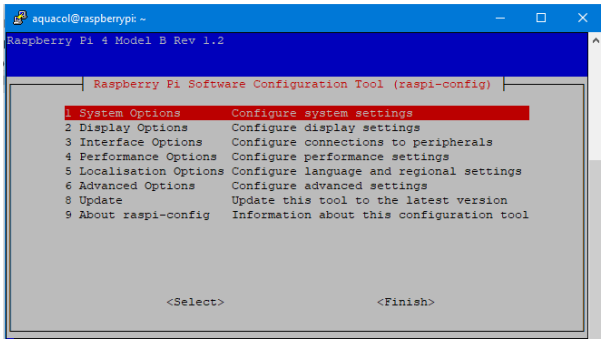


Figura 79 - Herramienta de configuración de Raspberry Pi

Tras la configuración se debe reiniciar el sistema para que se apliquen los cambios.

En VNC Viewer se crea una nueva conexión incluyendo la dirección IP de la placa. Llegados a este punto se podrá acceder a la Raspberry mediante el escritorio remoto utilizando los credenciales de acceso, como se muestra en las siguientes figuras:

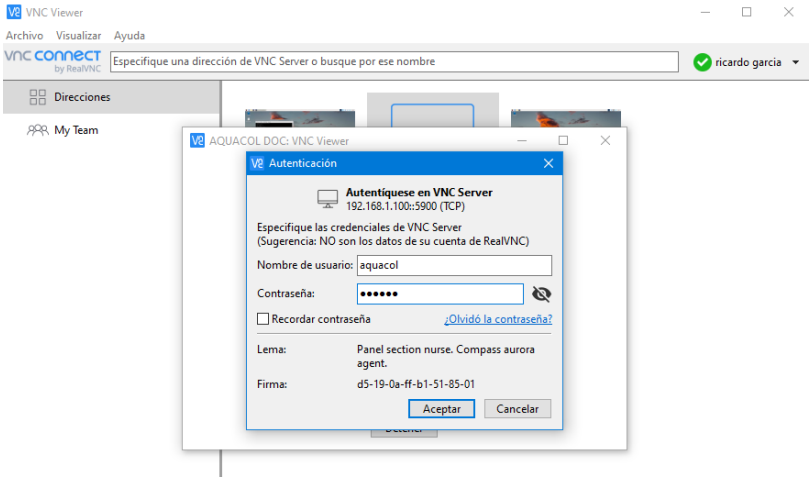


Figura 80 – Autenticación de acceso a Raspberry Pi con VNC Viewer

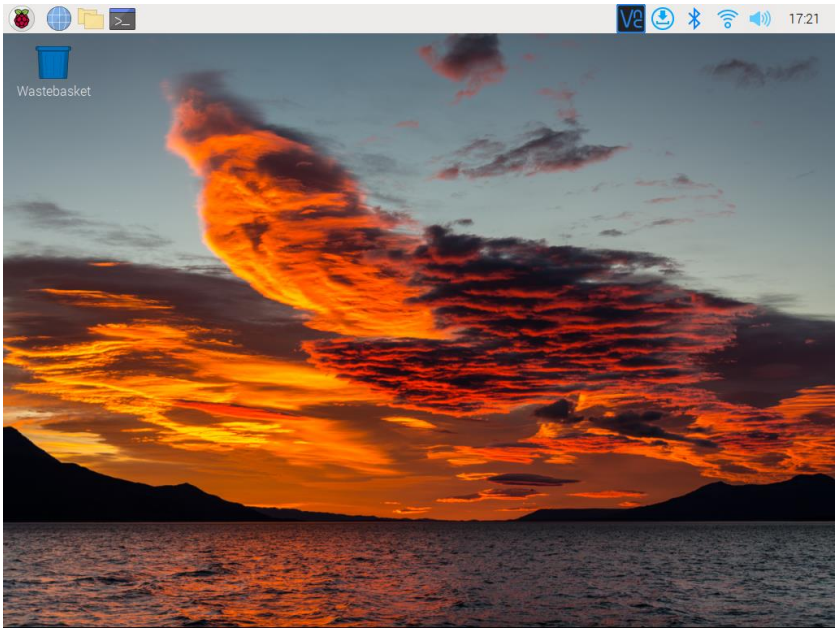


Figura 81 – Escritorio Raspberry Pi

Una vez dentro se actualiza el sistema operativo ejecutando los siguientes comandos:

>> sudo apt-get update

>> sudo apt-get upgrade

4.2.3.3. IP estática

Es conveniente establecer una dirección IP estática para la Raspberry dentro de la red local. De esta manera se podrá utilizar siempre la misma conexión en VNC y se evitan problemas de acceso en caso de que el servidor DHCP cambie la IP de la placa.

Para ello se accede al fichero de configuración del servidor DHCP mediante el editor de texto “nano”, utilizando el siguiente comando:

>> sudo nano /etc/dhcpcd.conf

Y se añade lo siguiente:

```
# Example static IP configuration:
#interface eth0
#static ip_address=192.168.0.10/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1

interface wlan0
static ip_address=192.168.1.103/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1 8.8.8.8
```

Figura 82 – Configuración de IP estática en archivo dhcpcd.conf

Siendo:

static ip_address: Dirección IP estática que se va a establecer

static routers: Puerta de enlace del router (gateway)

static domain_name_servers: Direcciones de servidores DNS

Tras salvar el fichero se reinicia el sistema para aplicar los cambios y se comprueba la nueva configuración:

```
aquacol@raspberrypi:~ $ ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.103  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::7cb9:fe27:2e06:cd6c  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:af:01:e4  txqueuelen 1000  (Ethernet)
        RX packets 220  bytes 33665 (32.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 257  bytes 36478 (35.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figura 83 – IP estática configurada correctamente

4.2.3.4. Instalación de paquetes

Como último paso de la configuración inicial, se instalan los siguientes módulos:

Pip

Sistema de gestión de paquetes utilizado para instalar y administrar módulos software de terceros en Python. Aunque viene incluido por defecto en la version completa del SO, se instala ejecutando el siguiente comando en un terminal:

>> sudo apt-get install python3-pip

Python3

Viene incluido en la version completa del SO. En caso de no contar con el, se instala con el siguiente comando:

>> sudo apt-get install python3 python3-dev

Psycopg2

Módulo que proporciona una interfaz de conexión y manipulación de bases de datos PostgreSQL desde Python. Se utiliza para establecer la conexión con la base de datos del proyecto, almacenando en ella la información obtenida de la instalación acuapónica.

Se instala utilizando pip:

>> pip install psycopg2

Paho.mqtt

Biblioteca de código abierto desarrollada por Eclipse que permite trabajar con clientes MQTT en distintos lenguajes de programación. En este proyecto se utiliza para implementar la comunicación entre la Raspberry Pi y un broker MQTT mediante un script de Python.

También se instala utilizando pip:

>> pip install paho-mqtt

4.2.4. Broker Mosquitto

Mosquitto es un broker MQTT de código abierto desarrollado por Eclipse. Un broker MQTT es un servidor encargado de gestionar la comunicación entre clientes MQTT, actuando como intermediario entre los dispositivos que publican mensajes y los dispositivos que los reciben. En este proyecto se ha utilizado para implementar la comunicación entre el ESP32 y la Raspberry Pi.

En primera instancia se utilizó un broker remoto con la siguiente dirección y puerto:

broker_address = test.mosquitto.org

broker_port = 1883

Finalmente se tomó la decisión de alojarlo en la Raspberry Pi de forma local, cuyo proceso de instalación se muestra a continuación:

1. Instalar el paquete de Mosquitto y clientes Mosquitto mediante el administrador de paquetes APT:

>> sudo apt install -y mosquito mosquito-clients

2. Configurar el servicio mosquitto para que se inicie automáticamente al iniciar la Raspberry:

>> sudo systemctl enable mosquitto.service

3. En la versión de Mosquitto 2.0 y posteriores, la configuración que viene por defecto no permite la conexión con clientes remotos (dispositivos que no sean la Raspberry Pi). Para habilitar el acceso remoto se debe modificar al archivo de configuración:

>> sudo nano /etc/mosquitto/mosquito.conf

Estableciendo el puerto 1883 para la recepción de mensajes y permitiendo la conexión de clientes sin autenticar:

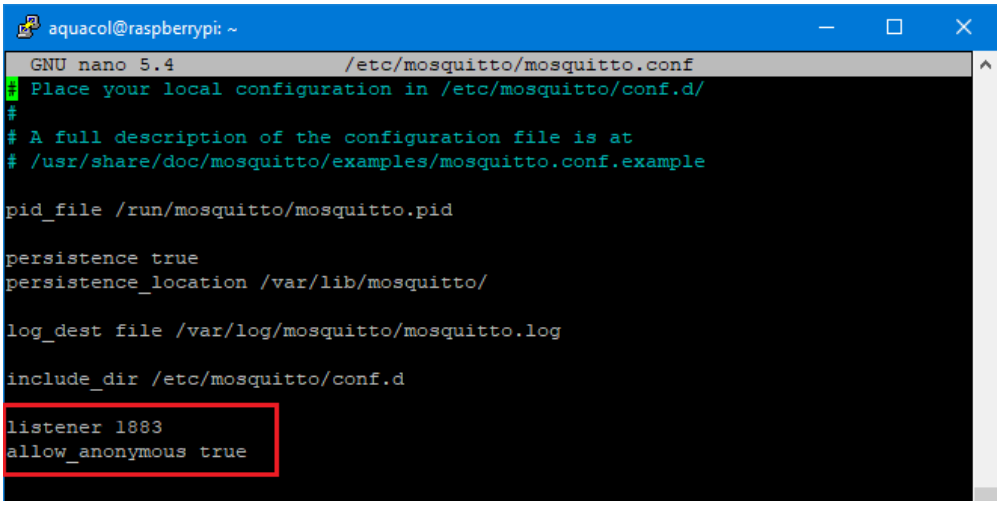


Figura 84 – Configuración bróker Mosquitto. Habilitar conexiones remotas

4. Reiniciar Mosquitto para aplicar los cambios:

>> sudo systemctl restart mosquitto

5. Si se quiere incluir autenticación para el acceso, que es recomendable, se añade un nombre de usuario y contraseña. En este caso se utilizan los mismos credenciales de acceso que posee la Raspberry:

>> sudo mosquitto_passwd -c /etc/mosquitto/passwd aquacol

Tras ejecutar el comando anterior con el nombre de usuario deseado, se pide que ingrese una constraseña dos veces.

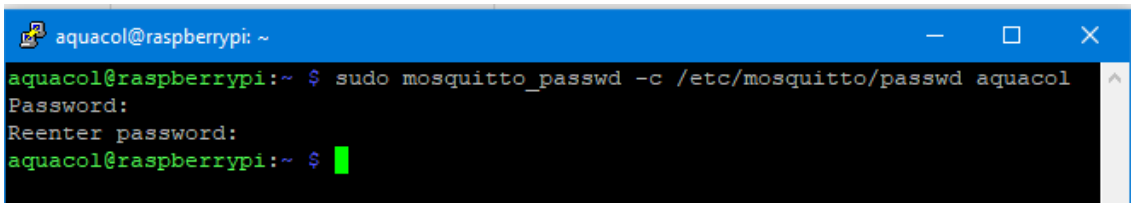


Figura 85 - Configuración bróker Mosquitto. Incluir autenticación (1)

Se creará un archivo de contraseña llamado "*passwd*" en el directorio "*/etc/mosquitto*".

6. Para incluir la autenticación en el archivo de configuración se incluyen las siguientes líneas en el orden que se muestra:

>> sudo nano /etc/mosquitto/mosquito.conf

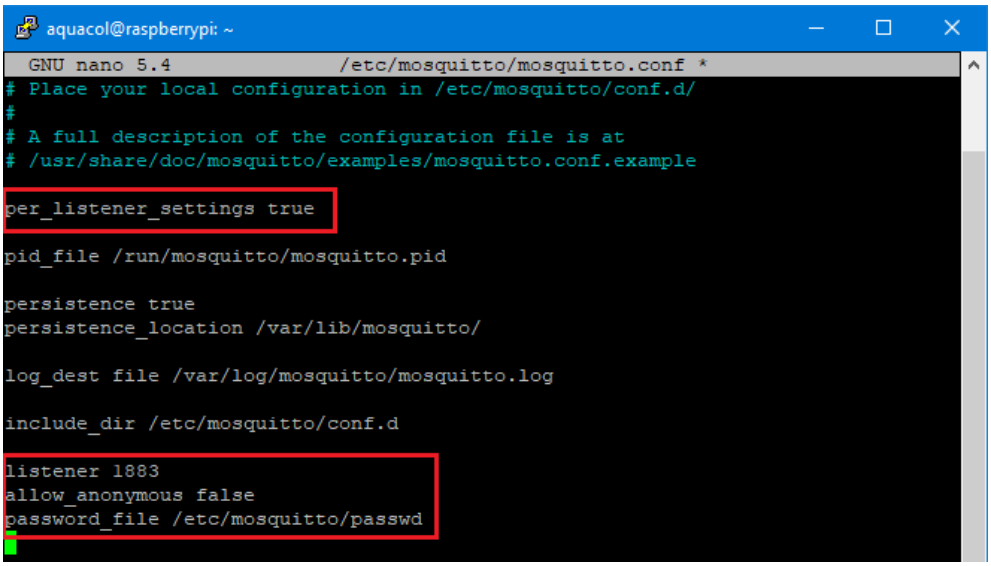


Figura 86 - Configuración bróker Mosquitto. Incluir autenticación (2)

7. Reiniciar Mosquitto para aplicar los cambios:

>> sudo systemctl restart mosquitto

8. Comprobar que Mosquitto se está ejecutando correctamente:

>> sudo systemctl status mosquitto

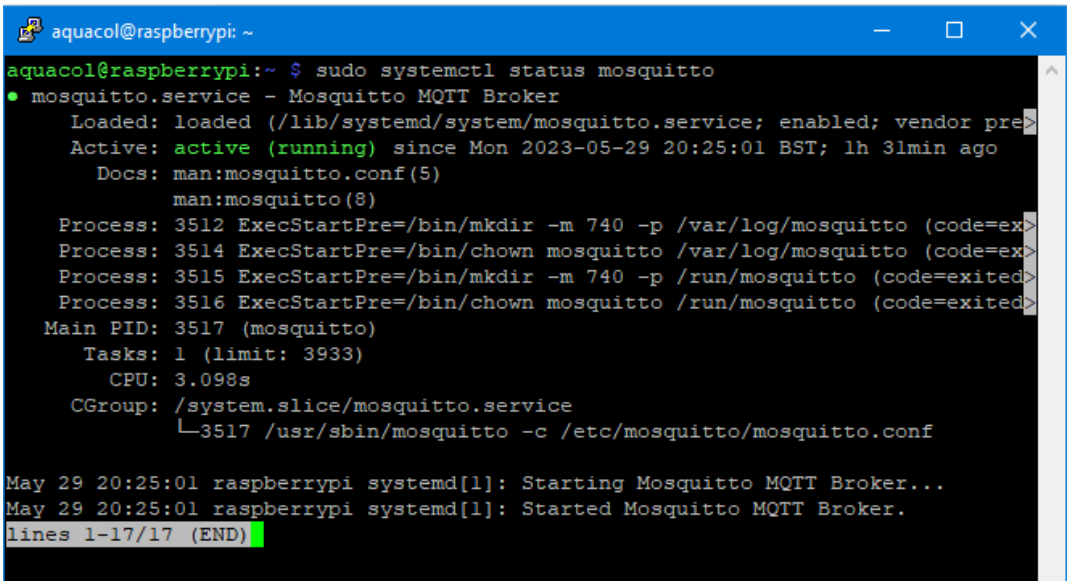


Figura 87 – Comprobar el estado del servicio mosquitto

Queda instalado y configurado el broker Mosquitto de manera local en la Raspberry Pi. A partir de ahora el ESP32 podrá enviar los datos de la instalación el broker utilizando la dirección IP de la Raspberry y el Puerto 1883.

4.2.5. Base de datos

La solución tomada fue almacenar los datos de la instalación en una base de datos PostgreSQL en la nube, mediante el proveedor de servicios ElephantSQL

ElephantSQL es un proveedor de bases de datos en la nube especializado en ofrecer servicios de bases de datos PostgreSQL. Se encarga de la infraestructura y el mantenimiento de ésta, ofreciendo una plataforma segura y disponible.

El primer paso es obtener una base de datos en la nube gratuita con ElephantSQL:

1. Acceder a: https://www.elephantsql.com/ y crearse una cuenta de usuario:

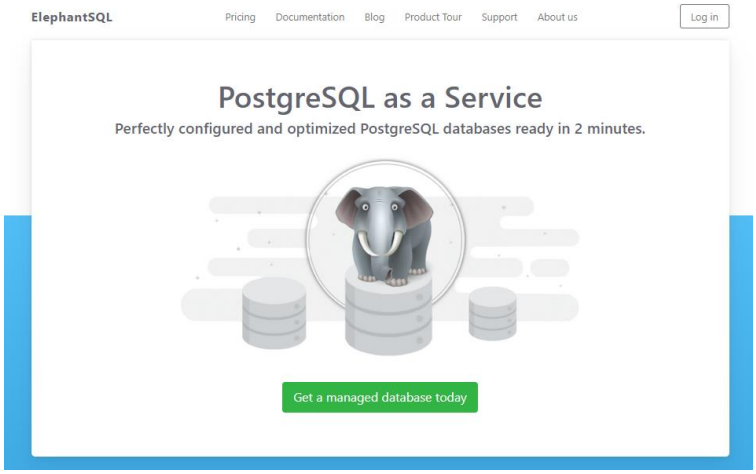


Figura 88 – Página web ElepehantSQL

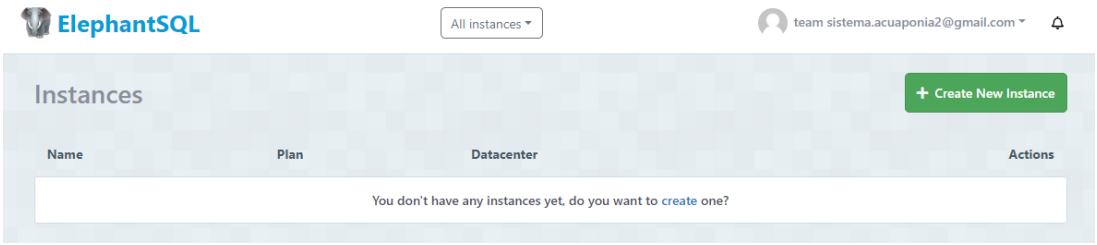


Figura 89 – Listado de instancias vacías tras crear una nueva cuenta

2. Crear una nueva instancia seleccionando:

- Nombre de la instancia/base de datos (aquaponic_system)
- Plan (Tiny Turtle, free)
- Centro de datos donde estará alojado el servidor (Google Compute Engine europe-west2 (London))

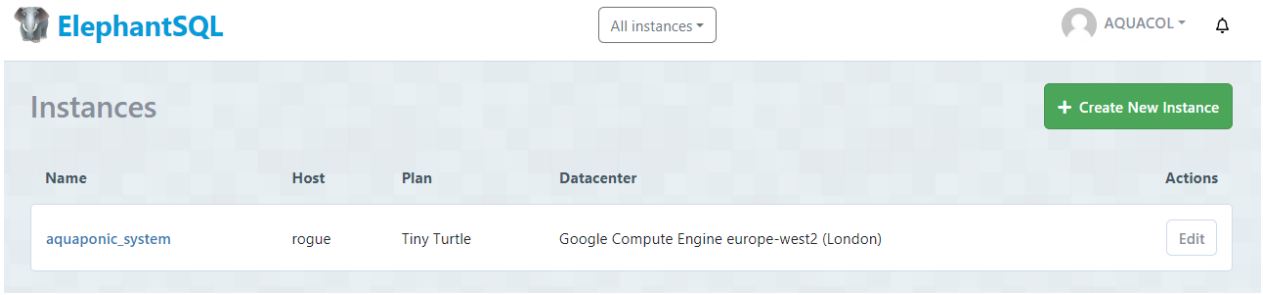


Figura 90 – Nueva instancia creada (aquaponic_system)

A continuación, se muestra la información que se debe conocer de la base de datos para poder establecer la conexión con ella más adelante:

- Servidor o host: rogue.db.elephantsql.com
- Usuario: vkpgnbux
- Base de datos: vkpgnbux
- Contraseña

Dicha información se encuentra disponible a través de la pestaña “DETAILS” dentro de la instancia creada:

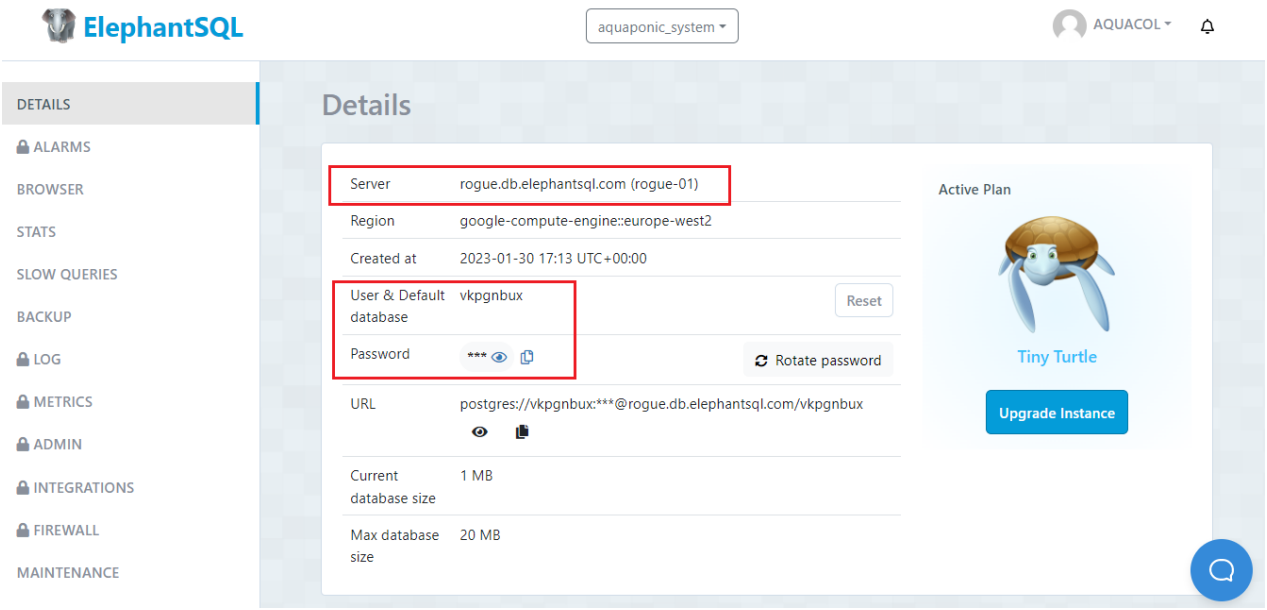


Figura 91 – Credenciales de la base de datos (instancia)

Para visualizar los datos existen dos alternativas:

1. Utilizar el navegador que ofrece ElephantSQL, a través de la pestaña “BROWSER”. Es la opción más cómoda, aunque la funcionalidad es limitada.

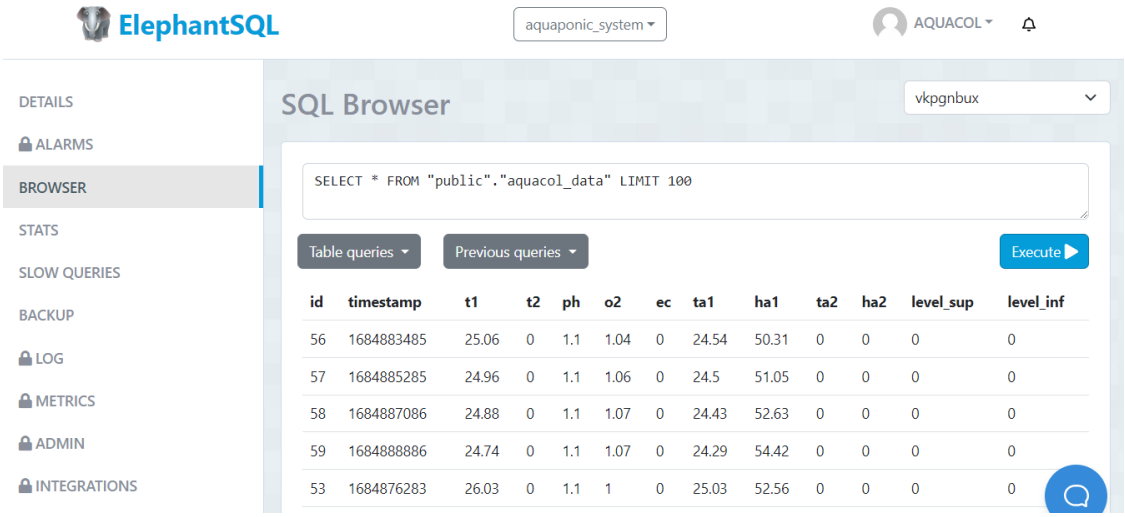


Figura 92 – Contenido de la tabla mostrada a través de ElephantSQL

2. Utilizar el administrador de bases de datos PostgreSQL PgAdmin4. Es la opción más completa, obteniendo todas las ventajas que brinda la herramienta en comparación con la opción anterior. En este caso es necesario acceder a la base de datos en PgAdmin4 utilizando la información descrita anteriormente.

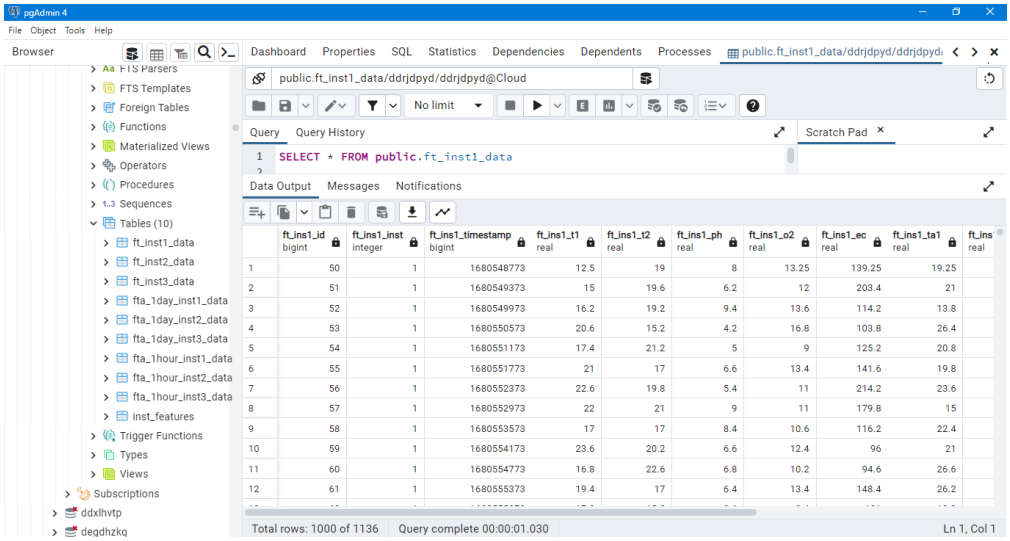


Figura 93 - Contenido de la tabla mostrada a través de PgAdmin4

4.2.6. Programa de almacenamiento de datos (Raspberry Pi + Python)

Con Mosquitto instalado y una base de datos creada, se describe el proceso de recepción y almacenamiento de datos. La solución consiste en un script en Python alojado en la Raspberry que utiliza el broker Mosquitto y la API psycopg2 con dos objetivos:

1. Recepción de mensajes MQTT desde el ESP32 alojado en la instalación.
2. Almacenamiento de dicha información en una tabla de la base de datos en la nube.

4.2.3.1. Librerías

A continuación, se describe cada una de las librerías necesarias para el programa de almacenamiento de datos en Python:

Nombre	**Descripción**
Paho.mqtt.client	Implementa el protocolo MQTT en Python, proporcionando una API para que los clientes se conecten a un broker MQTT con el fin de publicar o suscribirse a determinados mensajes a través de un topic.
Psycopg2	Adaptador de base de datos que permite la interacción de Python con bases de datos PostgreSQL, proporcionando una interfaz eficiente y fácil de usar para realizar operaciones de consulta y manipulación de datos.
Threading	Biblioteca propia de Python que permite crear y administrar hilos (threads) en un programa, permitiendo la ejecución simultánea de tareas.

Tabla 10 – Descripción de librerías Python del código de almacenamiento de datos

4.2.3.2. Descripción de funciones

En la siguiente tabla se muestran las funciones que se utilizaron para implementar la recepción y almacenamiento de datos de la instalación, indicando a qué hilo pertenecen:

Hilo	Función	Descripción
Thread_MQTT	On_connect	Función callback proporcioada por la librería paho.mqtt.client. Se invoca cuando el cliente MQTT establece una connexion exitosa con el broker. Una vez establecida la conexión, el cliente se suscribe al topic.
	On_message	Función callback proporcioada por la librería paho.mqtt.client. Se invoca cuando el cliente MQTT recibe un mensaje en el topic al que está suscrito, decodificándolo y levantando una bandera que indica la recepción de un nuevo mensaje para almacenar en la base de datos.
Thread_DB	DB_conexion	Función encargada de establecer una conexión con la base de datos PostgreSQL utilizando la librería “psycopg2”.
	Insert_data	Función encargada de realizar una consulta con la base de datos para inserter la última trama recibida por MQTT.
	Create_table	Función opcional encargada de crear una nueva tabla en la base de datos, configurando el nombre y el tipo de dato de cada columna.

Tabla 11 - Descripción de funciones del código de almacenamiento de datos

4.2.3.3. Descripción del código

La funcionalidad del código se describe mediante el siguiente diagrama de flujo:

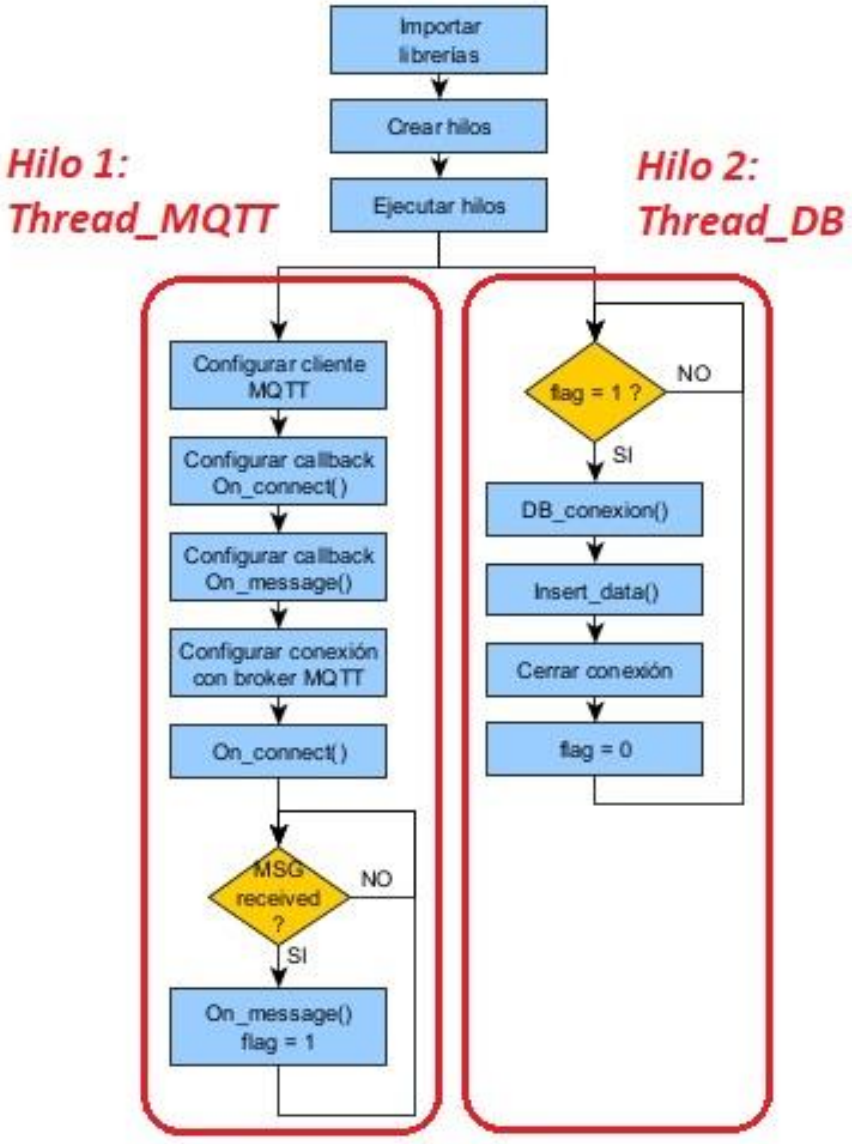


Figura 94 – Diagrama de flujo del código de almacenamiento de datos

4.2.3.4. Ejecución automática del programa

Como último paso, se configura el script para que sea lanzado automáticamente al encender la Raspberry sin necesidad de hacer loggin. Para ello se utiliza “systemd”, un sistema de inicio y administración de servicios utilizado en distribuciones de Linux. Se encarga de iniciar, detener y administrar los servicios del sistema de manera eficiente.

A continuación, se describen los pasos a realizar para la configuración:

1. Alojar el script en una ruta conocida. En este caso: */home/aquacol*

El programa se llama *receive_data.py*

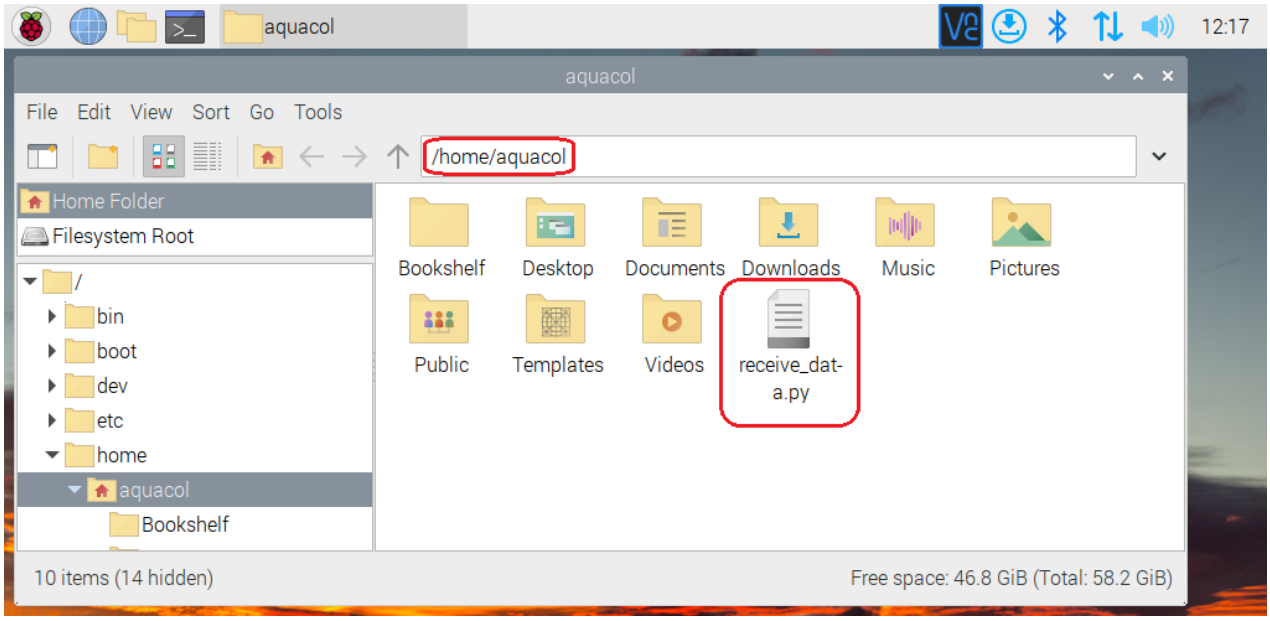


Figura 95 – Código Python de almacenamiento de datos en Raspberry Pi (receive_data.py)

2. Crear un archivo de configuración (archivo de unidad) que informa a systemd de lo que hay que hacer y cuándo:

>> sudo nano /lib/system/system/receive_data.service

```
 GNU nano 5.4          /lib/systemd/system/receive_data.service
[Unit]
Description=Receive Data Service
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python /home/aquacol/receive_data.py
WorkingDirectory=/home/aquacol
User=aquacol

[Install]
WantedBy=multi-user.target
```

Figura 96 – Configuración del servicio para la ejecución automática del código de almacenamiento de datos (receive_data.py)

3. Habilitar permisos para el servicio:

>> sudo chmod 644 /lib/systemd/system/receive_data.service

4. Configurar systemd para que inicie el servicio automáticamente tras la secuencia de arranque:

>> sudo systemctl daemon-reload

>> sudo systemctl enable receive_data.service

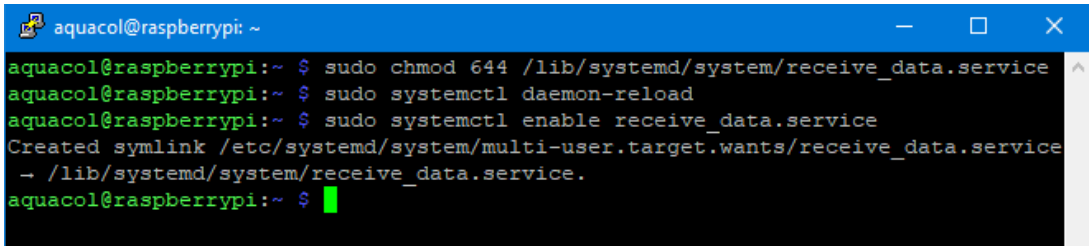


Figura 97 – Habilitar servicio receive_data.service

5. Reiniciar el sistema para aplicar los cambios

>> sudo reboot

6. Comprobar que el servicio se encuentra activo tras iniciar el sistema:

>> sudo systemctl status receive_data.service

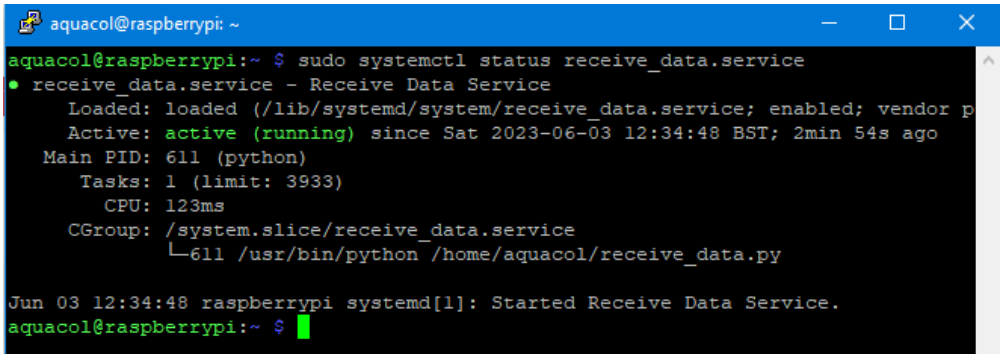


Figura 98 - Comprobar el estado del servicio receive_data.service

Llegados a este punto queda correctamente configurada la Raspberry para lanzar automáticamente el script de almacenamiento de datos al encender la placa.

5. RESULTADOS

En este apartado se exponen los resultados obtenidos durante el desarrollo del proyecto. Se muestran imágenes de la instalación acuapónica con el sistema de monitorización integrado, desde las primeras pruebas con la placa perfoda hasta el prototipo final en PCB.

También se muestran las lecturas de la instalación almacenadas tanto en la microSD como en la base de datos, incluyendo la evolución de diferentes parámetros a lo largo de un día.

5.1. Instalación acuapónica

En las siguientes imágenes aparece la instalación hidropónica y el tanque de peces propio del cultivo acuícola.

Figura 99 – Cultivo hidropónico (1)

Figura 100 – Cultivo acuícola. Tanque de peces

Figura 101 - Cultivo hidropónico (2)

En este caso se muestra el prototipo final desplegado en la instalación acuapónica.

Figura 102 – Instalación con prototipo Aquacol en placa perforada (1)

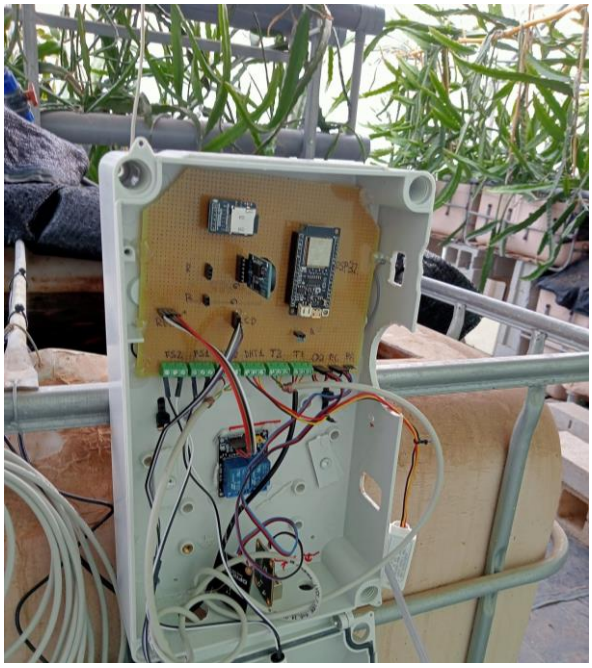


Figura 103 - Instalación con prototipo Aquacol en placa perforada (2)

Figura 104 – Instalación con prototipo Aquacol en PCB (1)

Figura 105 - Instalación con prototipo Aquacol en PCB (2)

5.2. Adquisición y almacenamiento de datos

5.2.1. Almacenamiento local

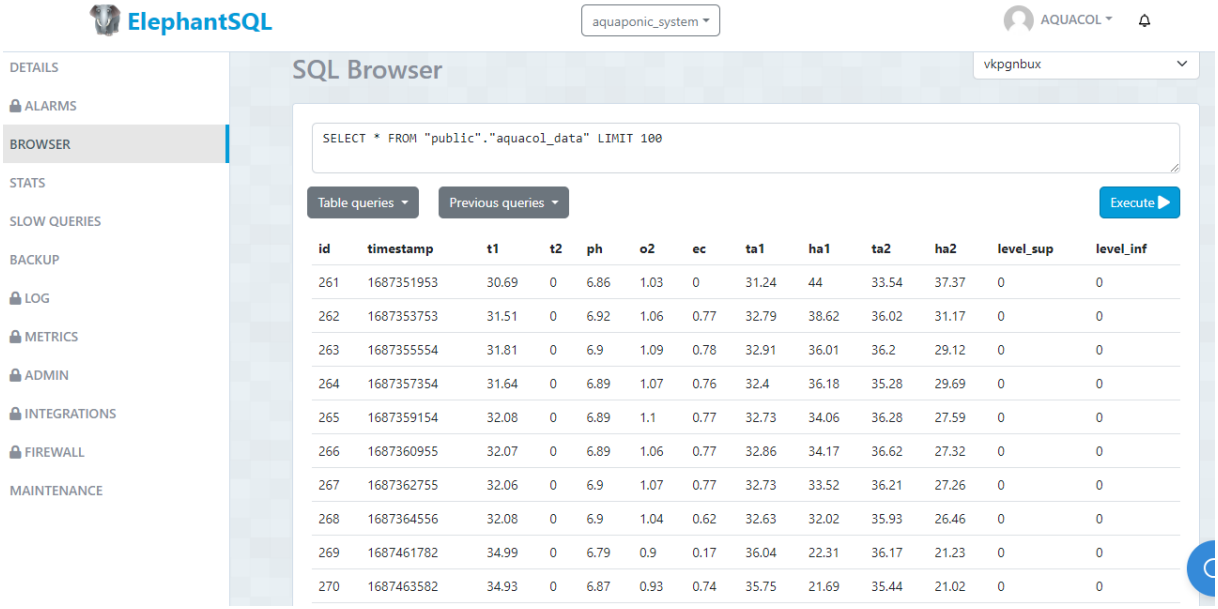
A continuación se muestra un fragmento del archivo CSV de la microSD encargado de almacenar las lecturas de forma local. Las medidas se toman cada media hora.

	A	B	C	D	E	F	G	H	I	J	K	L
1	TimeStamp	Temp1	Temp2	pH	Oxigeno	EC	Ta1	Ha1	Ta2	Ha2	Level_Sup	Level_Inf
2	1685732900	25.90	0.00	5.52	1.22	0.00	25.32	54.83	0.00	0.00	0	1
3	1685735147	26.03	0.00	6.96	1.19	0.83	25.40	53.39	0.00	0.00	0	1
4	1685736948	26.07	0.00	6.85	1.19	0.82	25.47	53.66	0.00	0.00	0	1
5	1685880922	33.08	0.00	5.86	0.96	0.44	35.39	26.29	38.61	21.95	0	0
6	1685882722	34.45	0.00	6.05	0.92	0.66	37.15	25.26	39.38	21.77	0	0
7	1685884523	35.65	0.00	6.07	0.88	0.44	38.26	23.64	41.54	19.90	0	0
8	1685886323	35.66	0.00	6.10	0.88	0.52	38.68	23.08	41.99	19.10	0	0
9	1685888123	36.11	0.00	6.14	0.86	0.65	38.88	22.55	41.56	19.26	0	0
10	1685889923	36.31	0.00	6.16	0.86	0.65	39.35	22.00	41.17	19.21	0	0
11	1685891723	36.04	0.00	6.18	0.87	0.66	39.40	21.78	41.51	18.81	0	0
12	1685893524	36.51	0.00	6.12	0.86	0.19	39.55	21.77	40.67	19.64	0	0
13	1685895324	35.51	0.00	6.18	0.88	0.61	38.92	21.61	37.57	21.56	0	0
14	1685897124	35.98	0.00	6.19	0.87	0.66	38.54	22.28	37.96	21.57	0	0
15	1685898924	37.02	0.00	6.11	0.84	0.04	38.76	22.44	39.21	21.16	0	0
16	1685900724	36.63	0.00	6.20	0.85	0.65	38.59	22.30	38.64	21.33	0	0
17	1685902525	35.69	0.00	6.13	0.88	0.17	37.29	23.79	37.47	22.73	0	0
18	1685904325	33.22	0.00	6.18	0.97	0.67	34.23	27.91	34.13	27.13	0	0
19	1685906125	32.15	0.00	6.18	1.02	0.71	32.48	30.24	32.37	29.38	0	0
20	1685907926	30.74	0.00	6.14	1.05	0.38	30.83	32.74	30.57	31.84	0	0
21	1685909726	29.96	0.00	6.19	1.12	0.74	29.74	33.89	29.49	32.95	0	0
22	1685911526	29.09	0.00	6.18	1.15	0.58	28.91	34.22	28.62	33.32	0	0
23	1685913326	28.16	0.00	6.20	1.21	0.77	27.90	35.97	27.59	35.17	0	0
24	1685915127	27.46	0.00	6.16	1.23	0.39	27.07	37.49	26.66	36.75	0	0
25	1685916927	27.02	0.00	6.21	1.28	0.77	26.65	39.18	26.32	38.39	0	0
26	1685918727	25.72	0.00	6.21	1.35	0.71	25.66	44.77	25.17	44.76	0	0
27	1685920527	23.46	0.00	6.33	1.40	0.27	23.76	60.80	22.59	64.22	0	0
28	1685922327	22.57	0.00	6.76	1.41	0.88	23.09	74.56	21.82	76.69	0	0
29	1685924128	22.59	0.00	6.91	1.41	0.88	22.59	78.54	21.46	79.04	0	0
30	1685925928	22.96	0.00	6.78	1.37	0.47	22.76	77.88	21.71	78.27	0	0
31	1685927728	22.31	0.00	6.46	1.41	0.84	22.56	78.00	21.44	77.97	0	0
32	1685929529	22.70	0.00	6.29	1.39	0.83	22.67	79.08	21.64	79.44	0	0
33	1685931329	22.06	0.00	6.46	1.45	0.85	22.00	79.98	21.00	80.48	0	0
34	1685933130	21.81	0.00	6.48	1.44	0.68	21.75	83.92	20.82	83.61	0	0
35	1685934930	21.32	0.00	6.71	1.47	0.88	21.36	82.59	20.45	82.17	0	0
36	1685936730	21.15	0.00	6.86	1.51	0.89	21.13	83.76	20.23	83.44	0	0
37	1685938531	20.77	0.00	7.06	1.53	0.92	20.81	85.66	20.01	84.96	0	0
38	1685940331	20.50	0.00	7.13	1.54	0.93	20.43	86.48	19.68	85.02	0	0
39	1685942132	20.28	0.00	7.09	1.55	0.92	20.21	84.34	19.50	82.90	0	0
40	1685943932	20.22	0.00	7.17	1.56	0.93	20.06	85.48	19.34	84.04	0	0
41	1685945733	20.06	0.00	7.30	1.57	0.94	20.00	86.92	19.25	85.83	0	0
42	1685947533	20.33	0.00	7.40	1.55	0.79	19.97	88.24	19.32	85.82	0	0
43	1685949333	20.77	0.00	7.24	1.52	0.93	20.47	85.86	19.69	84.36	0	0
44	1685951134	20.67	0.00	7.18	1.53	0.71	20.53	84.63	19.79	83.45	0	0
45	1685952934	20.83	0.00	7.31	1.47	0.28	20.76	86.88	19.89	86.02	0	0
46	1685954734	21.21	0.00	7.27	1.47	0.78	21.45	85.12	20.37	85.56	0	0
47	1685956534	22.13	0.00	7.07	1.40	0.45	22.37	81.59	21.18	82.50	0	0
48	1685958334	23.57	0.00	6.75	1.33	0.50	24.26	70.45	23.16	71.81	0	0
49	1685960134	24.10	0.00	6.46	1.33	0.82	25.17	60.57	24.52	61.01	0	0
50	1685961934	25.64	0.00	6.30	1.27	0.11	26.25	55.52	26.39	54.10	0	0
51	1685963734	27.22	0.00	6.26	1.37	0.78	28.37	46.48	28.79	43.94	0	0
52	1685965535	27.48	0.00	6.14	1.31	0.06	28.73	43.87	28.95	41.82	0	0
53	1685967335	27.67	0.00	6.12	1.37	0.08	28.89	43.21	28.96	41.40	0	0
54	1685969135	28.72	0.00	6.18	1.81	0.79	30.10	40.97	30.50	38.67	0	0
55	1685970935	30.51	0.00	6.17	1.56	0.74	32.44	38.04	33.55	34.57	0	0

Figura 106 – Lecturas almacenadas en un archivo CSV de la microSD

5.2.2. Almacenamiento en base de datos

En este caso se muestran las lecturas almacenadas en la base de datos remota, también tomadas cada media hora. Para su visualización se utiliza ElephantSQL.



id	timestamp	t1	t2	ph	o2	ec	ta1	ha1	ta2	ha2	level_sup	level_inf
261	1687351953	30.69	0	6.86	1.03	0	31.24	44	33.54	37.37	0	0
262	1687353753	31.51	0	6.92	1.06	0.77	32.79	38.62	36.02	31.17	0	0
263	1687355554	31.81	0	6.9	1.09	0.78	32.91	36.01	36.2	29.12	0	0
264	1687357354	31.64	0	6.89	1.07	0.76	32.4	36.18	35.28	29.69	0	0
265	1687359154	32.08	0	6.89	1.1	0.77	32.73	34.06	36.28	27.59	0	0
266	1687360955	32.07	0	6.89	1.06	0.77	32.86	34.17	36.62	27.32	0	0
267	1687362755	32.06	0	6.9	1.07	0.77	32.73	33.52	36.21	27.26	0	0
268	1687364556	32.08	0	6.9	1.04	0.62	32.63	32.02	35.93	26.46	0	0
269	1687461782	34.99	0	6.79	0.9	0.17	36.04	22.31	36.17	21.23	0	0
270	1687463582	34.93	0	6.87	0.93	0.74	35.75	21.69	35.44	21.02	0	0

Figura 107 – Información de la base de datos visualizada con ElephantSQL (1)

id	timestamp	t1	t2	ph	o2	ec	ta1	ha1	ta2	ha2	level_sup	level_inf
261	1687351953	30.69	0	6.86	1.03	0	31.24	44	33.54	37.37	0	0
262	1687353753	31.51	0	6.92	1.06	0.77	32.79	38.62	36.02	31.17	0	0
263	1687355554	31.81	0	6.9	1.09	0.78	32.91	36.01	36.2	29.12	0	0
264	1687357354	31.64	0	6.89	1.07	0.76	32.4	36.18	35.28	29.69	0	0
265	1687359154	32.08	0	6.89	1.1	0.77	32.73	34.06	36.28	27.59	0	0
266	1687360955	32.07	0	6.89	1.06	0.77	32.86	34.17	36.62	27.32	0	0
267	1687362755	32.06	0	6.9	1.07	0.77	32.73	33.52	36.21	27.26	0	0
268	1687364556	32.08	0	6.9	1.04	0.62	32.63	32.02	35.93	26.46	0	0
269	1687461782	34.99	0	6.79	0.9	0.17	36.04	22.31	36.17	21.23	0	0
270	1687463582	34.93	0	6.87	0.93	0.74	35.75	21.69	35.44	21.02	0	0
271	1687474385	27.85	0	6.85	1.27	0.82	26.93	42.35	26.64	41.44	0	0
272	1687794874	42.76	0	6.76	0.02	0.63	43.92	18.26	48.13	14.54	0	0
273	1687796674	42.45	0	6.78	3129.1	0.63	43.91	18.33	47.8	14.73	0	0
274	1688588881	33.72	0	6.36	0.93	0.63	33.97	24.71	33.92	23.72	0	0
275	1688590681	32.06	0	6.33	0.98	0.45	32.16	28.77	31.98	27.9	0	0
276	1688592481	30.46	0	6.29	1.03	0.04	30.31	31.43	30.03	30.55	0	0
277	1688594281	29.35	0	6.37	1.08	0.76	28.95	35.33	28.64	34.36	0	0
278	1688596082	28.43	0	6.37	1.11	0.77	27.86	36.94	27.5	36.15	0	0
279	1688597882	27.7	0	6.38	1.14	0.77	27.07	40.84	26.63	39.99	0	0
280	1688599683	26.96	0	6.39	1.17	0.79	26.37	48.82	25.91	48.03	0	0
281	1688601483	26.28	0	6.43	1.2	0.8	25.68	55.72	25.23	55.04	0	0
282	1688603284	25.77	0	6.48	1.23	0.81	25.15	59.74	24.68	59.41	0	0
283	1688605084	25.39	0	6.6	1.24	0.81	24.7	63.75	24.26	63.16	0	0
284	1688606885	24.99	0	6.57	1.26	0.82	24.28	61.5	23.85	61.29	0	0

Figura 108 - Información de la base de datos visualizada con ElephantSQL (2)

5.2.3. Evolución diaria de las lecturas

Para ilustrar la evolución de los parámetros de la instalación se graficaron las medidas obtenidas.a lo largo de un día completo, obervandose comportamientos razonables:

1. La temperatura del agua cae hasta los 24 grados a las 7:00am, alcanzando un máximo de 38 grados aproximadamente sobre las 6 de la tarde.

2. Como es lógico, con la temperatura ambiente ocurre algo similar, alcanzando su valor máximo unas horas antes con respecto a la temperatura de agua. Se observa la diferencia de temperatura entre el sensor ambiental alojado en el interior de la instalación (t1) y en el exterior (t2).

3. La humedad relativa presenta un comportamiento inverso al de la temperatura ambiente, obtiendo valores mínimos cuando la temperatura alcanza sus valores máximos. En este caso también se observa la diferencia de humedad entre el sensor ambiental alojado en el interior de la instalación (HR1) y en el exterior (HR2).

4. El valor del pH se va reduciendo a lo largo del día debido a la actividad de los peces y al aumento de la temperatura.

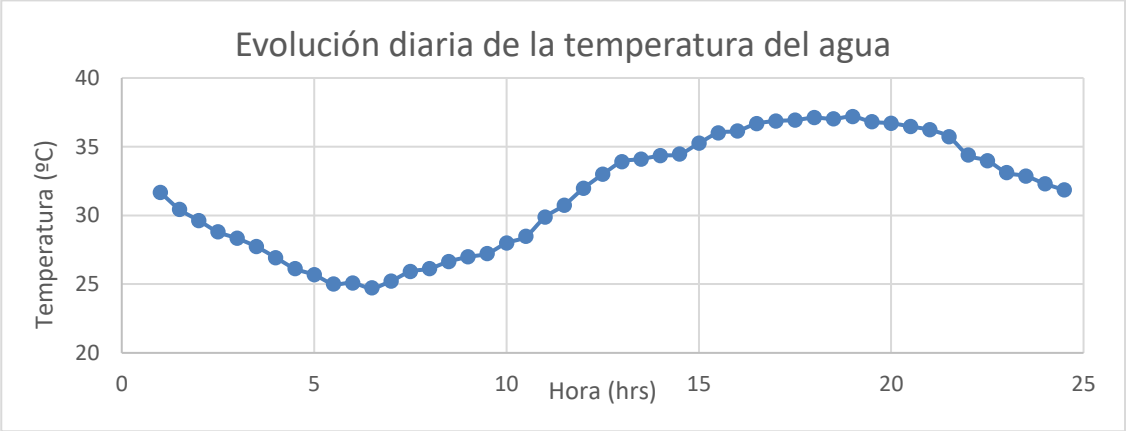


Figura 109 – Evolución diaria de la temperatura del agua en el tanque

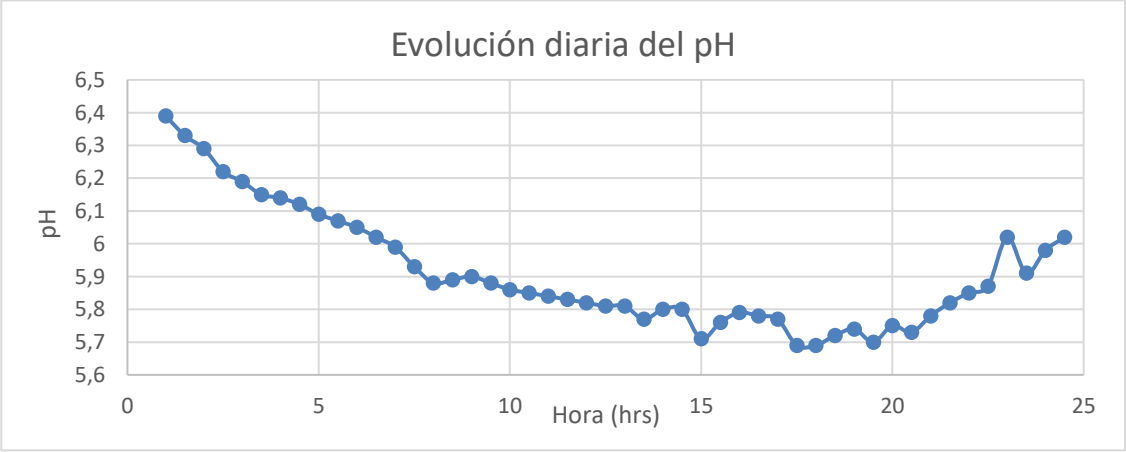


Figura 110 – Evolución diaria del pH

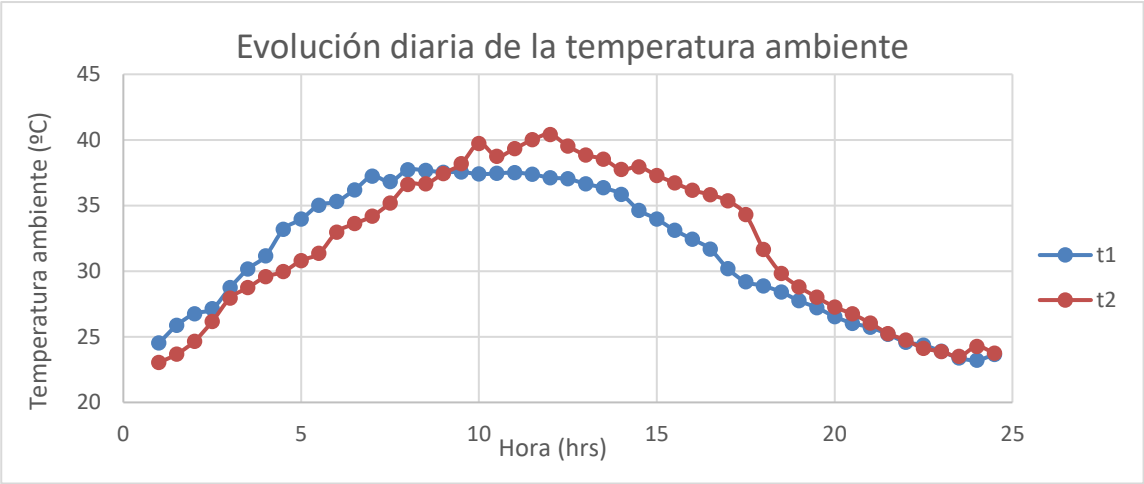


Figura 111 – Evolución diaria de la temperatura ambiente dentro y fuera de la instalación

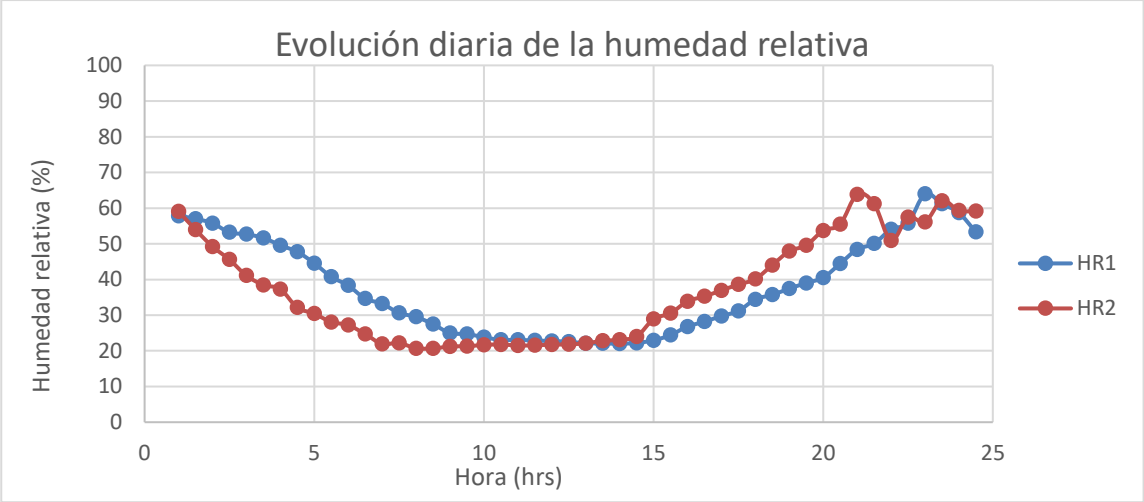


Figura 112 – Evolución diaria de la humedad ambiente dentro y fuera de la instalación

6. PROBLEMAS SURGIDOS

En este apartado se describen los problemas y limitaciones encontradas durante el desarrollo del proyecto, los cuales deberán ser tenidos en cuenta para futuras versiones del prototipo.

6.1. Convertidor analógico digital no lineal

El convertidor analógico-digital (ADC) del ESP32 presenta un comportamiento no lineal, lo que supone un problema a la hora de buscar precisión en las medidas.

ArduinoIDE cuenta con una librería específica para calibrar el ADC del ESP32 llamada "esp_adc_cal". Esta librería contiene funciones para corregir los errores en las medidas causados por las variaciones de los voltajes de referencia del ADC entre diferentes placas ESP32.

Este método implica caracterizar uno de los ADC con una atenuación determinada, obteniedo así una curva característica (curva de voltaje del ADC) que tenga en cuenta la diferencia en el voltaje de referencia.

Para ello se utiliza un tipo de variable propia de la librería, encargada de almacenar los parámetros del ADC (*esp_adc_cal_characteristics_t*). La curva característica se utiliza para convertir las lecturas del ADC a voltaje. Tiene la siguiente forma:

$$y = Ax + B$$

Donde:

y: Voltaje medido en mV

x: Lectura del ADC

A y B: Coeficientes

Para utilizar este método correctamente se deben seguir tres pasos:

1. Definir una variable tipo *esp_adc_cal_characteristics_t* que albergue los parámetros que caracterizan al ADC.

Ej:

esp_adc_cal_characteristics_t adc_chars

2. Caracterizar el ADC durante la configuración utilizando el método *esp_adc_cal_characterize:*

Ej:

esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12, 0, &adc_chars)

Donde:

- ADC_UNIT_1: Identificador del ADC a calibrar (ADC _1)

- ADC_ATTEN_DB_11: Atenuación que caracteriza al ADC (11dB)
- ADC_WIDTH_BIT_12: Resolución del ADC (12 bits)
- El cuarto argumento de entrada es el valor por defecto del voltaje de referencia (0V)
- El último argumento es un puntero a la dirección de memoria de la variable que almacena los parámetros característicos del ADC (&adc_chars)

3. Obtener el voltaje medido en base a la lectura del ADC y sus parámetros, utilizando la función *esp_adc_cal_raw_to_voltage*

Ej:
float sensor_raw = analogRead(sensor_pin);
float sensor_voltage = esp_adc_cal_raw_to_voltage(sensor_raw, &adc_chars)

Será necesario utilizar este método de calibración, o utilizar un ADC externo, al trabajar con microcontroladores ESP32 en aplicaciones que requieran cierta precisión en las lecturas de sensores analógicos.

6.2. Sensores analógicos

6.2.1. Sensor de pH (DFRobot SEN0169)

Inicialmente se utilizó el SEN0169 V1, presentando los siguientes problemas:

1. El procedimiento de ajuste (calibración) que aporta el fabricante no funciona correctamente. Fue necesario crear una calibración propia, obteniendo la pendiente y el offset del sensor mediante operaciones matemáticas.
2. Al no incluir regulador de tensión, cualquier cambio en la alimentación afecta considerablemente en la medida, generando la siguiete problemática:
    - Calibrar utilizando el PC no sirve de nada si luego se utilizará una fuente de alimentación diferente.
    - Cualquier variación del consumo afecta directamente a la medida, como por ejemplo otros sensores analógicos que se conecten.
3. No incluye corrección por temperatura.
4. El potenciómetro de la placa de acondicionamiento, encargado de ajustar la ganancia, afecta a la linealidad de la medida. Se ajustó su posición hasta lograr una respuesta lo más lineal posible.

En segunda instancia se integró el SEN0169 V2, presentando también varios problemas:

1. La librería del sensor está mal programada y no realiza la corrección de la temperatura.
2. El procedimiento de calibración que ofrece la librería exige dos pH fijos (4 y 7), impidiendo realizarse si se está utilizando la calibración del ADC. Fue necesario prescindir de la librería y crear una calibración propia, obteniendo la pendiente y el offset del sensor mediante operaciones matemáticas.

Con ambos sensores se obtienen medidas alteradas cuando el agua se encuentra en movimiento.

6.2.2. Sensor de conductividad eléctrica (DFR0300)

Se observó que el sensor de conductividad eléctrica DFR0300 devolvía lecturas nulas al operar durante largos periodos de tiempo. Tras leer detenidamente la documentación que ofrece el fabricante, indica que el sensor es una sonda de laboratorio y no se debe sumerjir durante mucho tiempo. Esto provoca que se acorte la vida de ésta.

Para futuras versiones será necesario adquirir un sensor de conductividad eléctrica capaz de funcionar ininterrumpidamente. La solución será buscar sondas industriales que ofrezcan buenas prestaciones a bajo coste.

6.3. Limitación del número de E/S digitales en ESP32

Durante el diseño PCB se incluyeron tres botones para dotar al sistema de una mayor funcionalidad, como realizar la calibración de los sensores a través de la pantalla LCD sin necesidad de acceder al código de ArduinoIDE.

BOTÓN	PIN
BUTTON_1	IO3
BUTTON_2	IO1
BUTTON_3	IO25

Tabla 12 - Distribución de pines para los botones en ESP32

Lo que no se tuvo en cuenta es que los pines IO3 y IO1 coinciden con los terminales de la comunicación serial (TX y RX), lo que impide cargar códigos en la placa si dichos puertos están ocupados. Fue necesario prescindir de los botones 1 y 2.

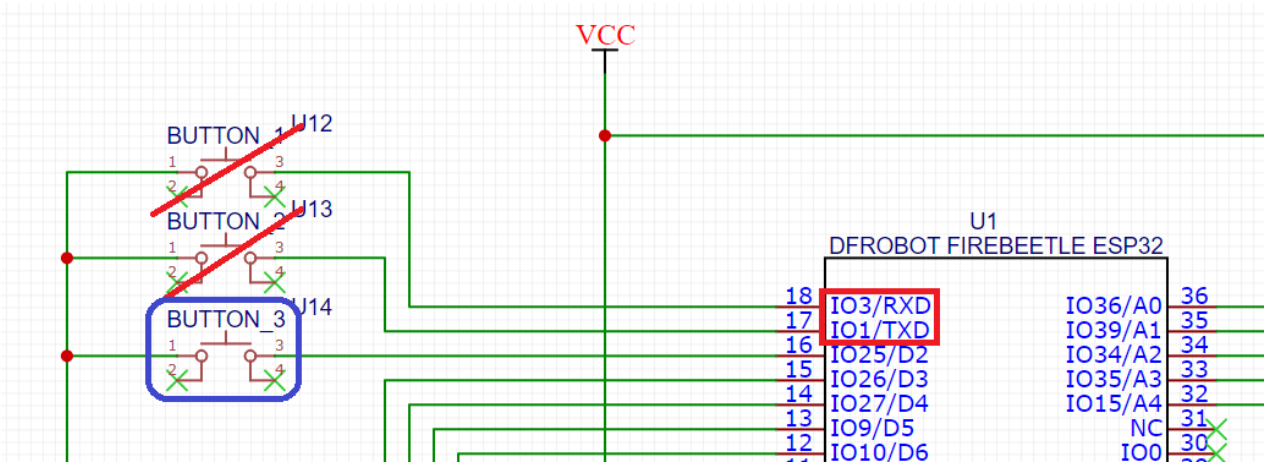


Figura 113 – Puertos Serial del ESP32 que no deben utilizarse

6.4. Raspberry Pi

Para la primera version del prototipo se utilizó una Raspberry Pi encargada de:

1. Alojar el broker MQTT Mosquitto

2. Gestionar la comunicación con la base de datos

Debido a las altas temperaturas que alcanza tanto el entorno de la instalación como la Raspberry Pi, fue necesario dotar a ésta de disipadores y ventilación activa, utilizando una carcasa con ventilador. Aún así, las altas temperaturas provocaban que la placa se apagase ocasionalmente, perdiendo la conexión con el broker y con la base de datos.

Para futuras versiones del prototipo será necesario alojar el broker y el código de gestión de la base de datos en Docker. De esta manera se evita la pérdida de conexión y se reduce el hardware utilizado, reduciendo así el costo total del sistema.

7. PRESUPUESTO

A continuación se muestra el coste total para el desarrollo del prototipo AQUACOL

Dispositivo	Modelo	Uds	Coste unitario (€)	Coste total (€)
Sensor de temperatura del agua	DS18B20	2	6,35	12,70
Sensor de pH	DFRobot SEN0169	1	61,43	61,43
Sensor de oxígeno disuelto	DFRobot SEN0237	1	155,38	155,38
Sensor de conductividad eléctrica	DFR0300	1	64,27	64,27
Sensor de nivel	Interruptor flotador RS PRO	2	27,38	54,76
Sensor ambiental	DHT22	2	7,64	15,28
ESP32	ESP32 FireBeetle Board	1	8,19	8,19
Kit Raspberry Pi	Raspberry Pi 4 B	1	225,99	225,99
Módulo microSD	DFRobot MicroSD Module V1.0	1	4,79	4,79
Reloj RTC	RTC DS3231	1	8,19	8,19
Display LCD	LCD I2C 2004A-V1.1	1	10,95	10,95
Relé	Songle 2CH relay module	1	5,99	5,99
PCB		1	1,87	1,87
Total				**629,79**

Tabla 13 - Presupuesto del prototipo AQUACOL

En caso de utilizar Docker y prescindir de la Raspberry Pi, el coste total del sistema se reduce a **403,80 euros.**

8. CONCLUSIONES

Se ha logrado desarrollar un sistema de monitorización y control de bajo coste para una instalación acuapónica utilizando dispositivos económicos y de buenas prestaciones que que permiten controlar el sistema de forma precisa, cumpliendo con los objetivos y requisitos establecidos en el proyecto.

A excepción de los problemas surgidos, se ha logrado una adquisición de datos en tiempo real, obteniendo lecturas precisas de los sensores. Además, se ha implementado un sistema de almacenamiento de datos en la nube utilizando un servicio de bases de datos gratuito.

Se ha optimizado el diseño del hardware permitiendo incluir todos los componentes y dispositivos en una única placa, incorporando una pantalla para monitorizar el estado de la instalación en tiempo real.

Este prototipo es la primera versión de un producto que podría promover y facilitar la implantación de cultivos acuapónicos de bajo coste en diferentes regiones.

REFERENCIAS

(CENADAC), P. C. (2015). Técnicas de acuaponía.

AEMA. (2020). Obtenido de https://aemahispanica.com/actualidad/tecnicas-hidroponicas/

Cáceres, G. P. (2021). *Tesis Doctoral. Caracterización y optimización de la producción de alimentos a través de sistemas acuapónicos a pequeña escala*.

Comparativa ESP32 y Arduino. (s.f.). Obtenido de elosciloscopio.com: https://elosciloscopio.com/comparacion-arduino-vs-esp8266-vs-esp32/

DFRobot. (s.f.). *Sensor de conductividad eléctrica DFR0300*. Obtenido de https://wiki.dfrobot.com/Gravity__Analog_Electrical_Conductivity_Sensor___Meter_V2__K=1__SKU_DFR0300

DFRobot. (s.f.). *Sensor de oxígeno disuelto SEN0237*. Obtenido de https://wiki.dfrobot.com/Gravity__Analog_Dissolved_Oxygen_Sensor_SKU_SEN0237

DFRobot. (s.f.). *Sensor pH SEN0169*. Obtenido de https://wiki.dfrobot.com/Analog_pH_Meter_Pro_SKU_SEN0169

Diego Ramírez, D. S. (2008). *LA ACUAPONÍA: UNA ALTERNATIVA ORIENTADA AL DESARROLLO SOSTENIBLE*. Obtenido de https://revistas.unimilitar.edu.co/index.php/rfcb/article/view/2230/1937

ESP32 ADC Tutorial. (s.f.). Obtenido de https://deepbluembedded.com/esp32-adc-tutorial-read-analog-voltage-arduino

Espressif. (s.f.). *Libraries and datasheets ESP32*. Obtenido de https://docs.espressif.com/projects/arduino-esp32/en/latest/libraries.html

García, S. (2021). *diariodelanzarote.com*. Obtenido de https://www.diariodelanzarote.com/noticia/la-discutida-acuicultura-en-lanzarote-una-oportunidad-con-muchas-alternativas

Hernández, L. D. (s.f.). *Sensor de temperatura DS18B20*. Obtenido de https://programarfacil.com/blog/arduino-blog/ds18b20-sensor-temperatura-arduino/

markerguides. (2022). Obtenido de https://www.makerguides.com/es/esp32-vs-arduino-speed-comparison/

Medidores y transmisores de oxígeno disuelto. (s.f.). Obtenido de https://www.es.endress.com/es/instrumentacion-campo/analisis-agua-liquidos-industria/oxigeno-disuelto-agua-medidor-saturacion

NAYLAM MECHATRONICS. (s.f.). Obtenido de https://naylampmechatronics.com/sensores-temperatura-y-humedad/58-sensor-de-temperatura-y-humedad-relativa-dht22-am2302.html

NAYLAM MECHATRONICS. (s.f.). *Módulo RTC DS3231*. Obtenido de https://naylampmechatronics.com/sensores/107-modulo-rtc-ds3231-eeprom-at24c32-i2c.html

Randomnerdtutorials. (2022). *Mosquitto broker on Raspberry Pi*. Obtenido de https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/

Raspberry Pi. (s.f.). *Autorun Python scripts using systemd*. Obtenido de https://www.raspberrypi-spy.co.uk/2015/10/how-to-autorun-a-python-script-on-boot-using-systemd/

Sensores de oxígeno. (2023). Obtenido de https://distron.es/sensor-oxigeno-caracteristicas/

University of Michigan. (s.f.). *Sensores de pH y viscosidad*. Obtenido de https://espanol.libretexts.org/Ingenieria/Ingenier%C3%ADa_Industrial_y_de_Sistemas/Libro%3A_Din%C3%A1mica_y_Controles_de_Procesos_Qu%C3%ADmicos_(Woolf)/03%3A_Sensores_y_Actuadores/3.07%3A_Sensores_de_pH_y_Viscosidad

Wikipedia. (s.f.). *Acuaponía*. Obtenido de https://es.wikipedia.org/wiki/Acuapon%C3%ADa

Wikipedia. (s.f.). *Hidroponía*. Obtenido de https://es.wikipedia.org/wiki/Hidropon%C3%ADa

1. Desarrollo hardware

1.1. Esquemático

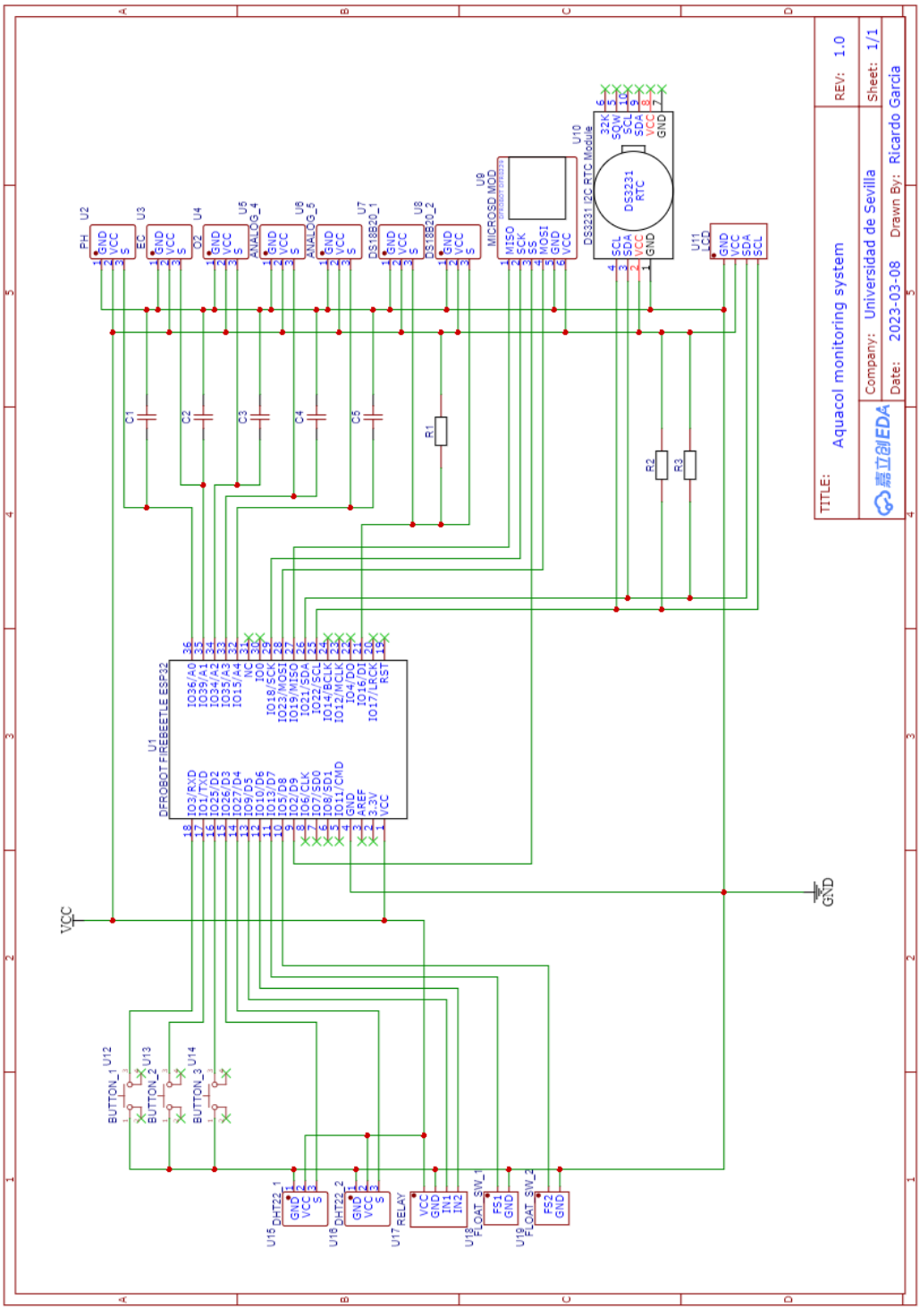


Figura 114 – Esquemático electrónico

1.2. Placa de circuito impreso (PCB)

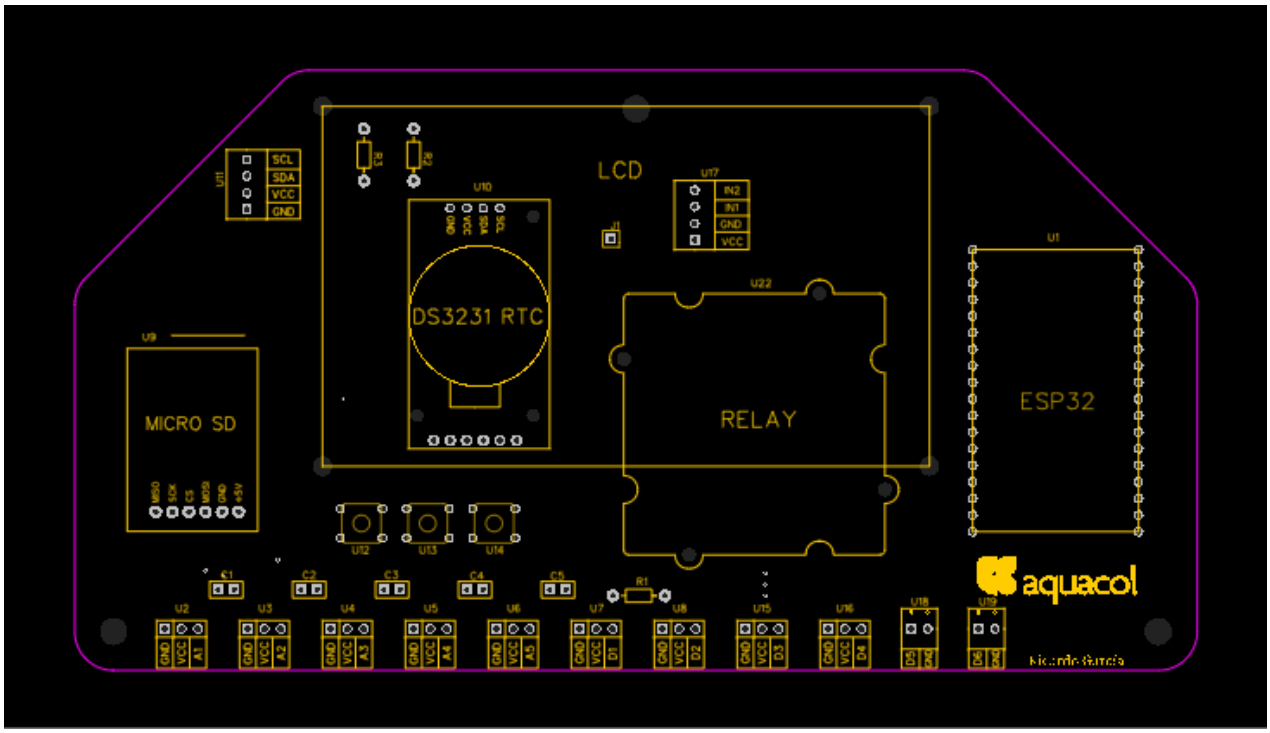


Figura 115 – Capa serigráfica superior de la PCB

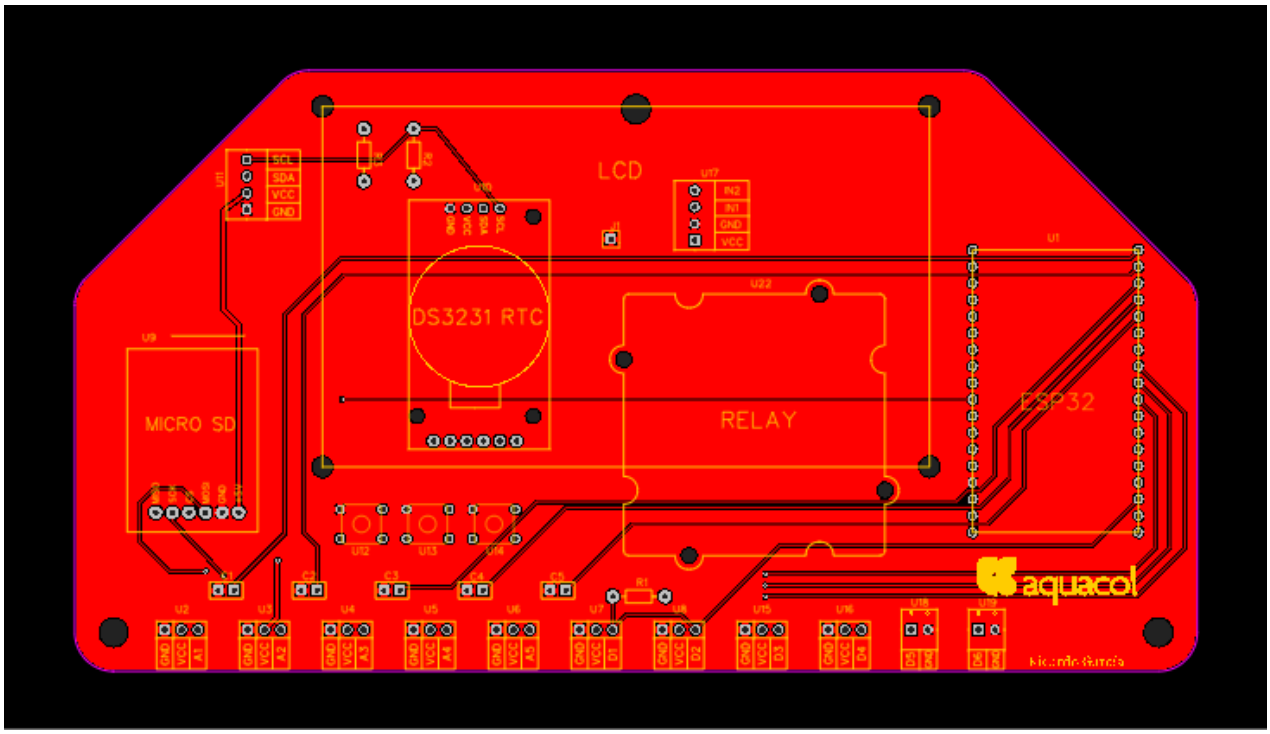


Figura 116 – Routing en capa superior de la PCB

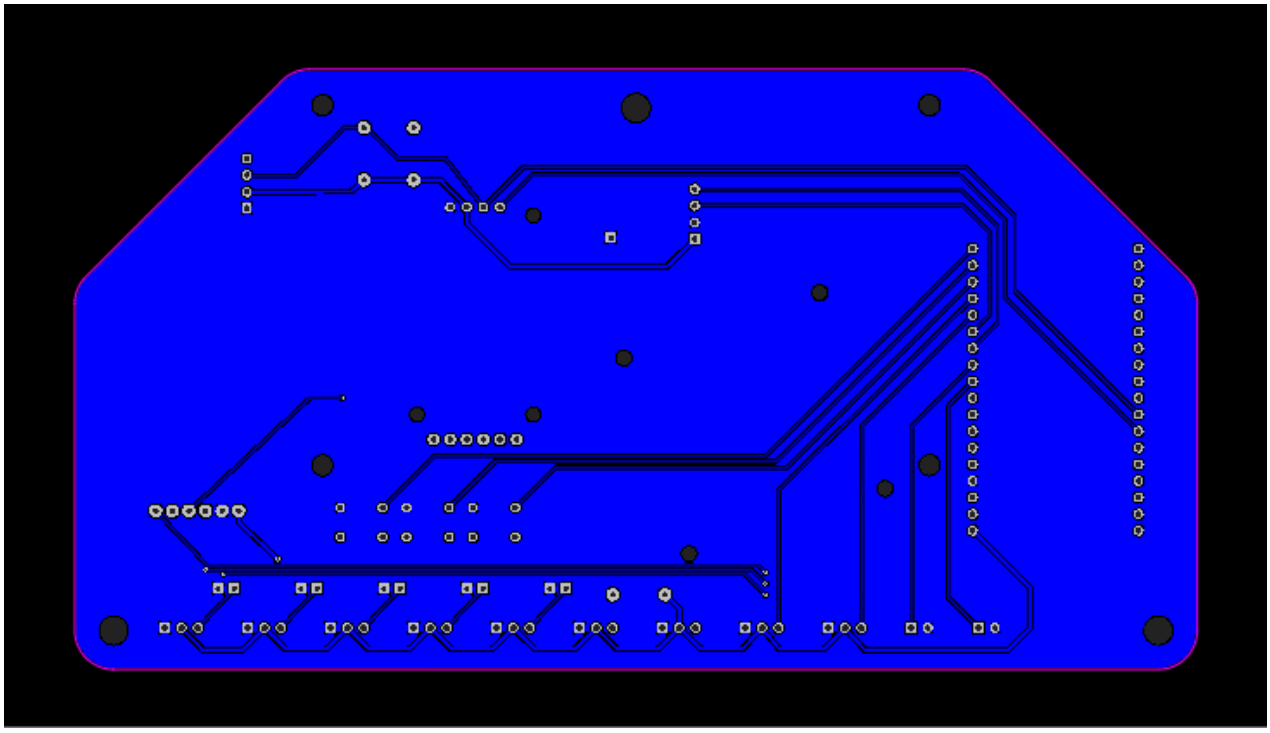


Figura 117 – Routing en capa inferior de la PCB

2. Desarrollo software

2.1. Código Arduino de adquisición de datos

```
// ***** IPORTACI?"N DE LIBRERÍAS ***** //
#include "Arduino.h"                  // Librería Arduino
[Arduino]
#include "esp_adc_cal.h"              // Librería de calibración del ADC
[Arduino]
#include "WiFi.h"                     // Librería para comunicación WIFI
[Hristo Gochkov]
#include "PubSubClient.h"             // Librería para comunicación MQTT
[Nicholas O'Leary]
#include "LiquidCrystal_I2C.h"        // Librería display LCD
[Frank de Brabander]
#include "EEPROM.h"                   // Librería memoria EEPROM
[Ivan Grokhotkov]
#include "OneWire.h"                  // Librería OneWire para los sensores
DS18B20       [Paul Stoffregen]
#include "DallasTemperature.h"        // Librería para lectura de sensores
DS18B20        [Miles Burton]
#include "DFRobot_ESP_EC.h"           // Librería para lectura sensor de
conductividad    [Mickael Lehoux (Greenponik)]
#include "RTClib.h"                   // Librería del reloj RTC
[Adafruit]
#include "DHT.h"                      // Librería para sensores ambientales
DHT22         [Adafruit]
```

```
#include "FS.h"                          // Librería para gestión de ficheros
[Hristo Gochkov, Ivan Grokhtkov]
#include "SD.h"                          // Libraría para gestión de memoria SD
[Arduino, SparkFun]
#include "SPI.h"                         // Librería para bus comunicación SPI
[Hristo Gochkov]

// ***** DEFINIC"N DE CONSTANTES ***** //
// Establecer pines
#define ph_pin 36                        // Pin sensor pH (analógico)
#define ec_pin 39                        // Pin sensor de conductividad electrica
(analógico)
#define o2_pin 34                        // Pin sensor oxigeno disuelto
(analógico)
#define dht22_pin1 26                    // Pin sensor DHT22 numero 1 (digital)
#define dht22_pin2 27                    // Pin sensor DHT22 numero 1 (digital)
#define float_sup_pin 13                 // Pin del sensor float switch superior
(digital)
#define float_inf_pin 5                  // Pin del sensor float switch inferior
(digital)
#define ev_pin 9                         // Pin del canal 1 del relé
#define bomb_pin 10                      // Pin del canal 2 del relé
#define onewire_pin 16                   // Pin de bus OneWire
// Parámetros del ADC y calibración
#define RESOL 4095                       // Resolución ADC ESP32 (12 bits)
#define VREF 3300                        // Tensión de trabajo del ADC ESP32
(3.3V)
#define MODE_CALIBRATION_O2 1            // MODE_CALIBRATION_O2 = 0 -->
Calibracion O2 a un punto / MODE_CALIBRATION = 1 --> Calibracion O2 a dos
puntos
#define CAL1_V (800)                     // Tensión de calibración primer punto
en mV (O2)
#define CAL1_T (15.75)                   // Temperatura de calibración primer
punto en °C (O2) [HIGH TEMPERATURE]
#define CAL2_V (630)                     // Tensión de calibración segundo punto
en mV (O2)
#define CAL2_T (5.20)                    // Temperatura de calibración segundo
punto en °C (O2) [LOW TEMPERATURE]
#define offset_ph 0.60                   // Offset para la medida del pH
// Periodos de ejecución de tareas y límites de tiempo
#define T_MQTT_ctrl 300000               // Periodo de envío del byte de control
para mantener la comunicación MQTT
#define T_adq 2500                       // Periodo de lectura de sensores
#define T_show 5000                      // Periodo de actualización del LCD
#define T_send 1800000                   // Periodo de almacenamiento de datos en
la SD y envío MQTT
#define T_level_ctrl 1000                // Periodo de ejecución del control de
nivel
#define T_ph_ctrl 60000                  // Periodo de ejecución del control de
Ph
#define T_1hour 3600000                  // Periodo para comprobar el nivel de pH
(control de pH) --> 1 hora
#define T_bomb_ON 600000                 // Periodo de tiempo activo para la
bomba (control de pH) --> 10 minutos
#define mqtt_conn_lim 15000              // Límite de tiempo de espera para la
conexión MQTT
#define wifi_conn_lim 15000              // Límite de tiempo de espera para la
conexión WiFi
#define MSG_BUFFER_SIZE    100           // Longitud máxima del búffer para envío
de trama por MQTT
#define Ph_ref 8.5                       // Referencia para el control de ph
```

```
// ***** GENERACI?"N DE OBJETOS ***** //
WiFiClient espClient;                  // Objeto de cliente wifi
PubSubClient client(espClient);        // Publicador subscriptor MQTT
LiquidCrystal_I2C lcd(0x27,20,4);      // Objeto para display LCD
DFRobot_ESP_EC ec;                     // Objeto para sensor de conductividad
RTC_DS3231 rtc;                        // Objeto para el reloj RTC
DHT dht_1(dht22_pin1, DHT22);          // Objeto para sensor ambiental DHT22
numero 1
DHT dht_2(dht22_pin2, DHT22);          // Objeto para sensor ambiental DHT22
numero 2
OneWire ourWire(onewire_pin);          // Objeto OneWire. Bus OneWire en pin 16
DallasTemperature DS18B20(&ourWire);   // Objeto para sensores de temperatura
DS18B20 en bus OneWire

// ***** DEFINICI?"N DE VARIABLES GLOBALES ***** //
unsigned long timestamp;                          // Marca de tiempo para la
adquisición de datos de sensores (almacenamiento en BD y SD)
unsigned long prev_time1, prev_time2, prev_time3;   // Variables para
almacenar instantes de tiempo anteriores
unsigned long prev_time4, prev_time5, prev_time6;
unsigned long prev_time7, prev_time8, prev_time9;
unsigned long prev_time10;
float Ta1;                     // Temperatura ambiente 1 (DHT22 1)
float Ha1;                     // Humedad ambiente 1     (DHT22 1)
float Ta2;                     // Temperatura ambiente 2 (DHT22 2)
float Ha2;                     // Humedad ambiente 2     (DHT22 2)
float Temp1;                   // Temperatura agua 1     (DS18B20 1)
float Temp2;                   // Temperatura agua 2     (DS18B20 2)
float pH;                      // Valor de pH
float ecc;                     // Valor de conductividad eléctrica
float o2;                      // Valor de oxígeno disuelto
int level_sup, level_inf;      // Estado de sensores float switch
int flag_LCD = 0;              // Bandera para alternar la impresión de los
datos en el LCD
int flag_PH = 0;               // Bandera que indica que la bomba del control
de pH está activa
int cont2 = 0;                 // Contador para el cálculo de valores medios

char ssid[50];                 // Nombre de la red wifi
char password[50];             // Clave de la red wifi
char lin = 0;                  // Bandera para lectura de los parámetros del
wifi
char caracter = 0;             // Carácter leído de los parámetros del wifi
uint8_t cont = 0;              // Contador para lectura de los parámetros del
wifi
int a = 0;                     // Bandera para indicar si ya se leyeron los
credenciales del WiFi de la SD

const char* mqtt_server = "192.168.1.105";          // Dirección IP del broker
local (Raspberry Pi)
//const char* mqtt_server = "broker.emqx.io";       // Broker remoto MQTT
("broker.emqx.io")
//const char* mqtt_server = "test.mosquitto.org";   // Broker remoto MQTT
("test.mosquitto.org")
char msg[MSG_BUFFER_SIZE];                          // Búffer de envío de
datos por MQTT
char ctrl_msg[] = "C";                              // Variable de control
para MQTT
```

```
esp_adc_cal_characteristics_t adc_chars;            // Variable para almacenar
las carcaterísticas del ADC
float Avg[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};         // Array de valores medios
de lectura de sensores

// ***** DEFINICI?"N DE FUNCIONES ***** //

////////////////////////////////
//// Envío de datos por MQTT ////
////////////////////////////////
void Send_MQTT(){
  // Todas las medidas se concatenan en un string, separándolas con una barra,
y se encapsulan en un array de tipo char ( msg[MSG_BUFFER_SIZE] )
  snprintf (msg, MSG_BUFFER_SIZE,
"%lu/%.2f/%.2f/%.2f/%.2f/%.2f/%.2f/%.2f/%.2f/%.2f/%d/%d", timestamp, Avg[0],
Avg[1], Avg[2], Avg[3], Avg[4], Avg[5], Avg[6], Avg[7], Avg[8], level_sup,
level_inf);
  // Se publica "msg" bajo el topic "aquacol"
  client.publish("aquacol", msg);
  Serial.println("MSG enviado por MQTT");
}
/////////////////////////////////////////////////////
//// Lectura de archivo de SD (Configuración WIFI) ////
/////////////////////////////////////////////////////
void readFile(fs::FS &fs, const char * path){
  Serial.printf("Reading file: %s\n", path);
  File file = fs.open(path);
  if(!file){
    Serial.println("Failed to open file for reading");
    return;
  }
  // Lectura del SSID y contraseña del WiFi del archivo de texto alojado en la
SD
  while(file.available()){
    caracter = (char)file.read();
    if(caracter == '/'){
      lin = 1;
      cont=0;
      continue;
    }
    if(lin == 0){
      ssid[cont] = caracter;
      cont++;
    }
    else{
      password[cont] = caracter;
      cont++;
    }
  }
  cont=0;
  file.close();
}
//////////////////////////////////////////////
//// Escritura de datos en archivo de SD ////
//////////////////////////////////////////////
void writeFile(fs::FS &fs, const char * path, String message){
  Serial.printf("Escribiendo en archivo: %s\n", path);
  File file = fs.open(path, FILE_WRITE);                    // Abrir el archivo
en modo escritura
  if(!file){                                                // Error abriendo
archivo. Salir de la función
```

```
    Serial.println("Fallo abriendo archivo");
    return;
  }
  if(file.println(message)){                              // Archivo abierto
correctamente. Escribir datos.
    Serial.println("Archivo escrito correctamente");      // Datos escritos
correctamente
  }
  else{
    Serial.println("Fallo al escribir en archivo");       // Fallo en
escritura de datos
  }
  file.close();
}
//////////////////////////////////////
//// Añadir datos en archivo de SD ////
//////////////////////////////////////
void appendFile(fs::FS &fs, const char * path, String message){
  Serial.printf("Appending to file: %s\n", path);
  File file = fs.open(path, FILE_APPEND);                     // Abrir el
archivo en modo añadir datos
  if(!file){                                                  // Error
abriendo archivo. Salir de la función
    Serial.println("Failed to open file for appending");
    return;
  }
  if(file.println(message)){                                  // Archivo
abierto correctamente. Añadir datos
    Serial.println("Message appended");                       // Datos
añadidos correctamente
  }
  else{
    Serial.println("Append failed");                          // Fallo
añadiendo datos
  }
  file.close();
}
//////////////////////////
//// Obtención del pH ////
//////////////////////////
float Get_pH_value(){
  float pH_raw = 0;                       // Valor de salida del ADC para el pH
(0-4095)
  float pH_voltage;                       // Tensión de salida del ADC para el
pH (mV)
  float pH_value;                         // Valor medido de pH
  int buf[10], t;                         // Buffer de 10 muestras de pH y
variable temporal

  for(int i = 0; i < 10; i++){            // Lectura de 10 valores
almacenándolos en array buf[]
    buf[i] = analogRead(ph_pin);          // Lectura señal analógica de pH
    delay(10);
  }
  for(int i = 0; i < 9; i++){             // Ordenar el array en orden
ascendente
    for(int j = i + 1; j < 10; j++){
      if(buf[i] > buf[j]){
        t = buf[i];
        buf[i] = buf[j];
        buf[j] = t;
```

```
      }
    }
  }
  for(int i = 2; i < 8; i++){                                          //
Eliminar los valores extremos y calcular la media
    pH_raw += buf[i];
  }
  pH_raw = pH_raw / 6;
  pH_voltage = esp_adc_cal_raw_to_voltage(pH_raw, &adc_chars)/1000.0;       //
Obtener la tensión en mV con el ADC calibrado
  pH_value = 3.5*pH_voltage + offset_ph;                               //
Convertir a pH
  return pH_value;                                                     //
Devuelve el valor medido de pH
}
/////////////////////////////////////////////////
//// Obtención de la conductividad eléctrica ////
/////////////////////////////////////////////////
float Get_ec_value(){
  float ec_voltage;                                 // Voltaje de salida del ADC
para el sensor de conductividad (EC)
  float ec_value;                                   // Valor medido de
conductividad (EC)
  ec_voltage = analogRead(ec_pin)*VREF/RESOL;       // Lectura voltaje salida
del ADC para EC
  ec_value =  ec.readEC(ec_voltage, Temp1);         // Conversión de voltaje en
EC con compensación de temperatura
  //ec.calibration(ec_voltage, tt);                 // Función de calibración
interna del sensor
  return ec_value;                                  // Devuelve el valor medido
de EC
}
/////////////////////////////////////////
//// Obtención del oxígeno disuelto ////
/////////////////////////////////////////
float Get_o2_value(){
  float V_saturation;
// Tensión de saturación
  const int DO_Table[41] = {
// Tabla de constantes para la obtención indirecta del oxigeno disuelto
    14460, 14220, 13820, 13440, 13090, 12740, 12420, 12110, 11810, 11530,
    11260, 11010, 10770, 10530, 10300, 10080, 9860, 9660, 9460, 9270,
    9080, 8900, 8730, 8570, 8410, 8250, 8110, 7960, 7820, 7690,
    7560, 7430, 7300, 7180, 7070, 6950, 6840, 6730, 6630, 6530, 6410};

  if(MODE_CALIBRATION_O2 == 0){
// Cálculo de la tensión de saturación (Calibración a 1 punto)
    V_saturation = (float)(CAL1_V + 35 * Temp1 - CAL1_T * 35);
  }
  else{
// Cálculo de la tensión de saturación (Calibración a 2 puntos)
    V_saturation = (float)((Temp1 - CAL2_T) * (CAL1_V - CAL2_V) / (CAL1_T -
CAL2_T) + CAL2_V);
  }

  float o2_raw = analogRead(o2_pin);
  float o2_voltage = esp_adc_cal_raw_to_voltage(o2_raw, &adc_chars);
// Obtener la tensión en mV que ofrece el ADC calibrado
  float o2_value = (float)(o2_voltage * (DO_Table[(int)Temp1]) /
V_saturation);   // Cálculo del oxígeno disuelto
```

```
  o2_value = o2_value / 1000.0;
// Pasar de mg/L
  return o2_value;
// Devuelve el valor calculado de oxígeno disuelto
}
////////////////////////////////////
//// Adquisición datos sensores ////
////////////////////////////////////
void Get_Sensors(){
  float aux;
  cont2++;
  // Filtrar lecturas para eliminar errores
  aux = dht_1.readTemperature();
  if(aux >= -20 && aux <= 60) {Ta1 = aux;}
  aux = dht_1.readHumidity();
  if(aux >= 0 && aux <= 100) {Ha1 = aux;}
  aux = dht_2.readTemperature();
  if(aux >= -20.0 && aux <= 60.0) {Ta2 = aux;}
  aux = dht_2.readHumidity();
  if(aux >= 0 && aux <= 100) {Ha2 = aux;}

  DS18B20.requestTemperatures();
  aux = DS18B20.getTempCByIndex(0);
  if(aux != -127.0) {Temp1 = aux;}
  aux = DS18B20.getTempCByIndex(1);
  if(aux != -127.0) {Temp2 = aux;}

  pH = Get_pH_value();                        // Lectura del pH
  ecc = Get_ec_value();                       // Lectura de la conductividad
eléctrica
  o2 = Get_o2_value();                        // Lectura del oxígeno disuelto
  level_sup = digitalRead(float_sup_pin);     // Lectura del sensor float switch
superior
  level_inf = digitalRead(float_inf_pin);     // Lectura del sensor float switch
inferior

  Avg[0] += Temp1;                            // Suma de medidas en array de
valores medios
  Avg[1] += Temp2;
  Avg[2] += pH;
  Avg[3] += o2;
  Avg[4] += ecc;
  Avg[5] += Ta1;
  Avg[6] += Ha1;
  Avg[7] += Ta2;
  Avg[8] += Ha2;
}
////////////////////////////////////////////////////////
//// Calcula el valor medio de las ultimas lecturas ////
////////////////////////////////////////////////////////
void Get_Average(){
  DateTime time = rtc.now();                // Obtiene el instante actual en
formate DateTime
  timestamp = time.unixtime();              // Convierte el dato anterior a
tiempo epoch (segundos transcurridos desde el 01/01/1970)

  for(int i = 0; i < 9; i++){               // Calcular valores medios de las
últimas lecturas (cont2)
    Avg[i] = Avg[i] / ((float)cont2);
    Serial.print(Avg[i]);
    Serial.print(" ");
```

```
  }
  cont2 = 0;
}
//////////////////////////////////
//// Actualizar datos en LCD ////
//////////////////////////////////
void Show_LCD(){
  lcd.clear();
  if (flag_LCD == 0){                          // Imprimir medidas de las
características del agua (LCD)
    lcd.setCursor(16,0); lcd.print("AGUA");
    lcd.setCursor(0,0); lcd.print("PH:"); lcd.print(pH, 2);
    lcd.setCursor(0,1); lcd.print("EC:"); lcd.print(ecc, 2); lcd.print("
ms/cm");
    lcd.setCursor(0,2); lcd.print("O2:"); lcd.print(o2, 2); lcd.print("
mg/L");
    lcd.setCursor(0,3); lcd.print("T1:"); lcd.print(Temp1, 2); lcd.print("C");
    lcd.setCursor(10,3); lcd.print("T2:"); lcd.print(Temp2, 2);
lcd.print("C");
    flag_LCD = 1;
  }
  else{                                        // Imprimir medidas ambientales
(LCD)
    lcd.setCursor(12,0); lcd.print("AMBIENTE");
    lcd.setCursor(0,2); lcd.print("Ta1:"); lcd.print(Ta1, 2); lcd.print("C");
    lcd.setCursor(11,2); lcd.print("H1:"); lcd.print(Ha1, 2); lcd.print("%");
    lcd.setCursor(0,3); lcd.print("Ta2:"); lcd.print(Ta2, 2); lcd.print("C");
    lcd.setCursor(11,3); lcd.print("H2:"); lcd.print(Ha2, 2); lcd.print("%");
    flag_LCD = 0;
  }
}
////////////////////////////////////////////////////////
//// Función para guardar las medidas en la memoria SD ////
////////////////////////////////////////////////////////
void Write_SD(){
  String data_str = "";                            // Genera string vacio
  data_str += String(timestamp) + ";";             // Concatena todas las
medidas
  data_str += String(Avg[0]) + ";";
  data_str += String(Avg[1]) + ";";
  data_str += String(Avg[2]) + ";";
  data_str += String(Avg[3]) + ";";
  data_str += String(Avg[4]) + ";";
  data_str += String(Avg[5]) + ";";
  data_str += String(Avg[6]) + ";";
  data_str += String(Avg[7]) + ";";
  data_str += String(Avg[8]) + ";";
  data_str += String(level_sup) + ";";
  data_str += String(level_inf);
  appendFile(SD, "/data_sensors.csv", data_str);  // Añade las medidas a
"data_sensors.csv" de la SD
}
////////////////////////////////////////////////////
//// Automatismo para el control de nivel de agua ////
////////////////////////////////////////////////////
void Level_Control(){
  level_sup = digitalRead(float_sup_pin);
  level_inf = digitalRead(float_inf_pin);
  if(level_sup == 0 && level_inf == 0){              // Depósito al nivel
máximo
```

```
    digitalWrite(ev_pin, HIGH);                          // Desactivar
electroválvula
    //Serial.println("Nivel MAXIMO");
  }
  else if(level_sup == 1 && level_inf == 1){             // Depósito al nivel
mínimo
    digitalWrite(ev_pin, LOW);                           // Activar
electroválvula
    //Serial.println("Nivel MINIMO");
  }
  else if(level_sup == 0 && level_inf == 1){             // Lectura no coherente.
ERROR
    Serial.println("ERROR EN LECTURA DE SENSORES");
    //Serial.println("LECTURA NO COHERENTE");
  }
}
//////////////////////////////////////////
//// Automatismo para el control de pH ////
//////////////////////////////////////////
void Ph_Control(){
  if(millis() - prev_time8 > T_1hour){                 // Si ha pasado una hora
    prev_time8 = millis();                             // Se guarda el instante
actual

    if (pH < Ph_ref){                                  // Si el valor de pH está
por debajo del umbral
      digitalWrite(bomb_pin, LOW);                     // Se activa la bomba
      prev_time9 = millis();                           // Se guarda el instante
en el que se activa la bomba
      flag_PH = 1;                                     // Se levanta bandera para
avisar de que la bomba está activa
    }
  }
  if(flag_PH == 1){                                    // Si la bomba está activa
    if(millis() - prev_time9 > T_bomb_ON){             // Se comprueba si ha
pasado el tiempo máximo de bomba activa
      digitalWrite(bomb_pin, HIGH);                    // Si es que si, se apaga
la bomba
      flag_PH = 0;                                     // Se baja la bandera
avisando de que la bomba ya se ha apagado
    }
  }
}
//////////////////////////////////////////////////////
//// Configuración e inicialización de dispositivos ////
//////////////////////////////////////////////////////
void setup_devices(){
  DS18B20.begin();                                     // Inicializa sensores de
temperatura DS18B20
  EEPROM.begin(32);                                    // Inicializa memoria
EEPROM
  rtc.begin();                                         // Inicializa el reloj RTC
  ec.begin();                                          // Inicializa sensor EC
  dht_1.begin();                                       // Inicializa sensor DHT22
número 1
  dht_2.begin();                                       // Inicializa sensor DHT22
número 2
  lcd.begin();                                         // Inicializa display LCD
  lcd.backlight();                                     // Reset display LCD
  pinMode(ev_pin, OUTPUT);                             // Establece el pin del
canal 1 del relé como salida digital (electroválvula)
```

```
  pinMode(bomb_pin, OUTPUT);                          // Establece el pin del
canal 2 del relé como salida digital (bomba de agua)
  digitalWrite(ev_pin, HIGH);                         // Inicialmente
electroválvula apagada
  digitalWrite(bomb_pin, HIGH);                       // Inicialmente bomba
apagada
  pinMode(float_sup_pin, INPUT_PULLUP);               // Establece el pin del
sensor float switch 1 como entrada digital
  pinMode(float_inf_pin, INPUT_PULLUP);               // Establece el pin del
sensor float switch 2 como entrada digital
  esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12, 0,
&adc_chars);   // Calibración del ADC

  lcd.clear();
  lcd.setCursor(0,0); lcd.print("Config devices...");
  delay(2000);
 }
////////////////////////////////////////////////
//// Configuración e inicialización del WIFI ////
////////////////////////////////////////////////
void setup_wifi(){
  lcd.clear();
  lcd.setCursor(0,0); lcd.print("Configurando WIFI...");
  delay(2000);
  if(a == 0){                                                 // Si aún no se
han obtenido los credenciales del WiFi:
    readFile(SD, "/conf_wifi.txt");                           // Leer archivo de
configuración del wifi para obtener nombre y clave
    a = 1;
  }
  if(ssid != 0 && password != 0){                             // Si se obtiene
nombre y clave del wifi:
    Serial.print("Conectando con: ");
    Serial.println(ssid);
    WiFi.mode(WIFI_STA);                                      // Configurar el
modo de la conexión
    WiFi.begin(ssid, password);                               // Iniciar la
conexión
    prev_time6 = millis();
    while (WiFi.status() != WL_CONNECTED){                    // Mientras no se
establezca la conexión:
      if (millis() - prev_time6 > wifi_conn_lim){             // Conteo de
tiempo. Si supera el tiempo límite: conexión no establecida
        lcd.setCursor(0,1); lcd.print(">>> Tiempo superado");
        lcd.setCursor(0,2); lcd.print(">>> WIFI ERROR");
        delay(2000);
        Serial.println("Tiempo superado");
        Serial.println("Conexión WiFi no establecida");
        return;                                               // Salir de la
función setup_wifi
      }
    }
    lcd.setCursor(0,1); lcd.print(">>> WIFI OK");
    delay(2000);
    randomSeed(micros());                                     // Conexión wifi
establecida correctamente
    Serial.println("");
    Serial.print("Conexion WiFi establecida con: ");
    Serial.println(ssid);
    Serial.println("Dirección IP: ");
```

```
    Serial.println(WiFi.localIP());                          // Mostrar
dirección IP asignada al ESP32
  }
  else{                                                              // Si no
se obtiene nombre y clave del wifi:
    lcd.setCursor(0,1); lcd.print(">>> Config error");
    lcd.setCursor(0,2); lcd.print(">>> WIFI ERROR");
    delay(2000);
    Serial.println("ERROR: Introducir nombre y clave WiFi en SD");  //
Conexión no establecida
    Serial.println("Conexión WiFi no establecida");
  }
}
//////////////////////////////////
//// Configuración memoria SD ////
//////////////////////////////////
void setup_sd(){
  lcd.clear();
  lcd.setCursor(0,0); lcd.print("Config SD...");
  delay(2000);
  if(SD.begin(2)){
    // Si no existe el archivo "data_sensors.csv" en la SD, se crea y se añade
la cabecera (nombres de parámetros)
    if(!SD.exists("/data_sensors.csv")){
      String header = "TimeStamp; Temp1; Temp2; pH; Oxigeno; EC; Ta1; Ha1;
Ta2; Ha2; Level_Sup; Level_Inf";
      writeFile(SD, "/data_sensors.csv", header);
    }
    lcd.setCursor(0,1); lcd.print(">>> SD OK");
    delay(2000);
    Serial.println("SD configurada correctamente");
  }
  else{
    lcd.setCursor(0,1); lcd.print(">>> SD ERROR");
    delay(2000);
    Serial.println("ERROR: SD no detectada");
  }
}
//////////////////////////////////////////
//// Configuración de comuniación MQTT ////
//////////////////////////////////////////
void setup_mqtt(){
  lcd.clear();
  lcd.setCursor(0,0); lcd.print("Config MQTT...");
  delay(2000);

  client.setServer(mqtt_server, 1883);                                //
Establecer broker y puerto MQTT
  Serial.println("Conectando con servidor MQTT...");
  prev_time7 = millis();
  while(!client.connected()){                                         //
Mientras no se establezca la conexión:

    String clientId = "ESP8266Client-";                               //
Asignar nuevo identificador de cliente
    clientId += String(random(0xffff), HEX);
    client.connect(clientId.c_str());                                 //
Conectar con cliente

    if(millis() - prev_time7 > mqtt_conn_lim){                        //
Conteo de tiempo
```

```
      lcd.setCursor(0,1); lcd.print(">>> Tiempo superado");
      lcd.setCursor(0,2); lcd.print(">>> MQTT ERROR");
      delay(2000);
      Serial.println("Tiempo superado. Conexion MQTT no establecida");  //
Tiempo superado. Conexión MQTT no establecida
      return;                                                            //
Salir de setup_mqtt
    }
  }
  client.loop();                                                         //
Conexión MQTT establecida correctamente
  lcd.setCursor(0,1); lcd.print(">>> MQTT OK");
  delay(2000);
  Serial.println("Conexion MQTT establecida");
}
////////////////////////////////////////////////
//// Iniciar instantes de tiempos anteriores ////
////////////////////////////////////////////////
void init_times(){
  prev_time1 = millis();    // Todas la variables de tiempo anterior se
inicializan con el tiempo actual
  prev_time2 = millis();
  prev_time3 = millis();
  prev_time4 = millis();
  prev_time5 = millis();
  prev_time6 = millis();
  prev_time7 = millis();
  prev_time8 = millis();
  prev_time9 = millis();
  prev_time10 = millis();
}
//////////////////////////////////////////
//// Configuración e inicializaciones ////
//////////////////////////////////////////
void setup(){
  Serial.begin(115200);       // Establecer baudrate para el puerto serie
  init_times();               // Inicizaliza las variables de instantes de
tiempo anterior
  setup_devices();            // Configuración de sensores y actuadores
  setup_sd();                 // Configuración tarjeta SD
  setup_wifi();               // Configuración de la conexión wifi
  setup_mqtt();               // Configuración de la conexión MQTT
}
//////////////////////////
//// Bucle infinito ////
//////////////////////////
void loop(){
  // Adquisición de datos de sensores
  if(millis() - prev_time1 > T_adq){
    prev_time1 = millis();
    Get_Sensors();
  }
  // Visualizar medidas a través del display LCD
  if(millis() - prev_time2 > T_show){
    prev_time2 = millis();
    Show_LCD();
  }
  // Almacenamiento de datos en SD y envío MQTT para BD
  if(millis() - prev_time3 > T_send){
    prev_time3 = millis();
    Get_Average();
```

```
    Write_SD();
    // Comprobar conexión WiFi y reconectar
    if(WiFi.status() != WL_CONNECTED){
      setup_wifi();
    }
    setup_mqtt();                          // Reconectar MQTT
    Send_MQTT();                           // Envío de datos por MQTT

    for(int i=0; i<9; i++){                // Resetear array de valores medios
      Avg[i] = 0;
    }
  }
  // Envío de byte de control por MQTT para mantener la conexión
  if(millis() - prev_time10 > T_MQTT_ctrl){
    prev_time10 = millis();
    client.publish("aquacol", ctrl_msg);
    Serial.println("control msg");
  }
  // Automatismo para el control de nivel de agua
  if(millis() - prev_time4 > T_level_ctrl){
    prev_time4 = millis();
    Level_Control();
  }
  // Automatismo para el control de pH
  if(millis() - prev_time5 > T_ph_ctrl){
    prev_time5 = millis();
    Ph_Control();
  }
}
```

2.2. Código Python de almacenamiento de datos

```
#!usr/bin/python3

########################################
# Importación de paquetes y librerías
import paho.mqtt.client as mqtt # Librería cliente MQTT
import threading                # Librería para ejecución de hilos en paralelo
import psycopg2                 # Librería de gestión de bases de datos
PostgreSQL

########################################
# Credenciales de la base de datos
data_base = "vkpgnbux"                              # Nombre de la base de datos
user = "vkpgnbux"                                   # Usuario
password = "rkrbRKe9yjudkRFA3dOU_t9ZGF2qd_mm"       # Contraseña
host = "rogue.db.elephantsql.com"                   # Dirección del host
table_name = "aquacol_data"                         # Tabla para almacenar los
datos de AQUACOL

########################################
# Datos para la conexión MQTT
client_id = "RaspberryPi"                           # Identificador del cliente
broker_address = "localhost"                        # Dirección del bróker local
(remotos: "broker.emqx.io" o "test.mosquitto.org")
broker_port = 1883                                  # Puerto de conexión
topic = "aquacol"                                   # Topic
```

```
########################################
# Definición de variables globales
global data          # Variable para almacenar las lecturas recibidas por MQTT
global msg_received  # Bandera que avisa de la llegada de un mensaje MQTT
msg_received = 0     # Inicialmente bajada

########################################
# Hilo de comunicación MQTT
def Thread_MQTT():
    # Función de conexión y suscripción al topic
    def on_connect(client, userdata, flags, rc):
        print("Conexion establecida. Cliente: ", client_id)
        client.subscribe(topic)

    # Función de interrupción para recepción de mensajes
    def on_message(client, userdata, message):
        global msg_received, data

        msg = message.payload.decode("utf-8") # Decodifica mensaje
        top = message.topic                   # Topic del mensaje

        if(len(msg) == 1):
            print("Control MSG")              # Mensaje de control
        else:

            # Definición de strings para almacenar lectura de sensores
            T1 = ''; T2 = ''; PH = ''; O2 = ''; EC = ''
            Ta1 = ''; Ha1 = ''; Ta2 = ''; Ha2 = ''; Tstamp = ''
            Lev_Sup = ''; Lev_Inf = ''
            cont = 0           # Contador para deserializar el mensaje
            # Deserialización del mensaje (lecturas separas por "/")
            for i in msg:
                if i == '/':    cont += 1
                elif cont == 0: Tstamp += i
                elif cont == 1: T1 += i
                elif cont == 2: T2 += i
                elif cont == 3: PH += i
                elif cont == 4: O2 += i
                elif cont == 5: EC += i
                elif cont == 6: Ta1 += i
                elif cont == 7: Ha1 += i
                elif cont == 8: Ta2 += i
                elif cont == 9: Ha2 += i
                elif cont == 10: Lev_Sup += i
                elif cont == 11: Lev_Inf += i
                else:           print('ERROR al deserializar trama MQTT')

            data = [Tstamp, T1, T2, PH, O2, EC, Ta1, Ha1, Ta2, Ha2, Lev_Sup,
Lev_Inf]     # Almacenar lecturas en array de strings
            for i in range (len(data)):     # Covertir en array de floats
                data[i] = float(data[i])
            msg_received = 1                # Aviso de llegada de mensaje MQTT
            print("\nTrama recibida: ")
            print(data)
            print("Topic: " + top)

    # Generar objeto cliente
    client = mqtt.Client(client_id)
    # Función de conexión con cliente y suscripción al topic
    client.on_connect = on_connect
```

```
    # Función de interrupción al recibir un mensaje
    client.on_message = on_message
    # Conexión con broker MQTT
    client.connect(broker_address, broker_port, keepalive=3600)
    # Bucle infinito
    client.loop_forever()


########################################
# Hilo de gestión de base de datos
def Thread_DB():
    global msg_received
    msg_received = 0

    # Función de conexión con la base de datos
    def DB_conexion(db, usr, passwd, host):
        conexion = psycopg2.connect(
            database=db,
            user=usr,
            password=passwd,
            host=host)
        cursor = conexion.cursor()
        return conexion, cursor

    # Función para crear la tabla necesaria en la base de datos
    def Create_table():                    # Generar consulta SQL (crear tabla)
        sql = "CREATE TABLE " + table_name + "(\
            ID SERIAL PRIMARY KEY,\
            TIMESTAMP bigint,\
            T1 real,\
            T2 real,\
            PH real,\
            O2 real,\
            EC real,\
            Ta1 real,\
            Ha1 real,\
            Ta2 real,\
            Ha2 real,\
            Level_sup int,\
            Level_inf int);"
        cursor.execute(sql)                # Ejecutar consulta SQL
        conexion.commit()

    # Función para insertar datos en la tabla
    def Insert_data():

        sql = "INSERT INTO " + table_name + "(TIMESTAMP, T1, T2, PH, O2, EC,
Ta1, Ha1, Ta2, Ha2, Level_sup, Level_inf)\
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);"
        cursor.execute(sql, data)          # Ejecutar consulta SQL
        conexion.commit()

    # Bucle infinito de recepción y almacenamiento de datos
    while(True):
        # Si llega un nuevo mensaje MQTT:
        if (msg_received == 1):
            # Abrir conexión con la base de datos
            conexion, cursor = DB_conexion(data_base, user, password, host)
            # Insertar lecturas en la tabla
            Insert_data()
            # Cerrar conexión
```

```
            conexion.close()
            # Aviso de mensaje recibido y almacenado
            msg_received = 0

############################
# Función principal
if __name__ == "__main__":

    # Crear hilo para la comunicación MQTT
    x = threading.Thread(target = Thread_MQTT)
    # Crear hilo para la conexión BD
    y = threading.Thread(target = Thread_DB)
    # Ejecutar hilo para la comunicación MQTT
    x.start()
    # Ejecutar hilo para la conexión BD
    y.start()
```